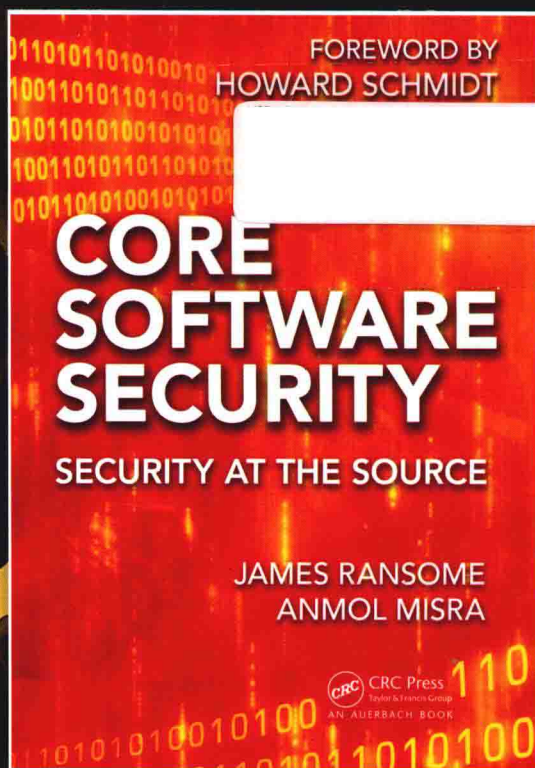


软件安全

从源头开始

[美] 詹姆斯·兰萨姆 (James Ransome) 著
安莫尔·米斯拉 (Anmol Misra)
丁丽萍 卢国庆 李彦峰 穆海蓉 曹原野 译
宋宇宇 审校

Core Software Security
Security at the Source



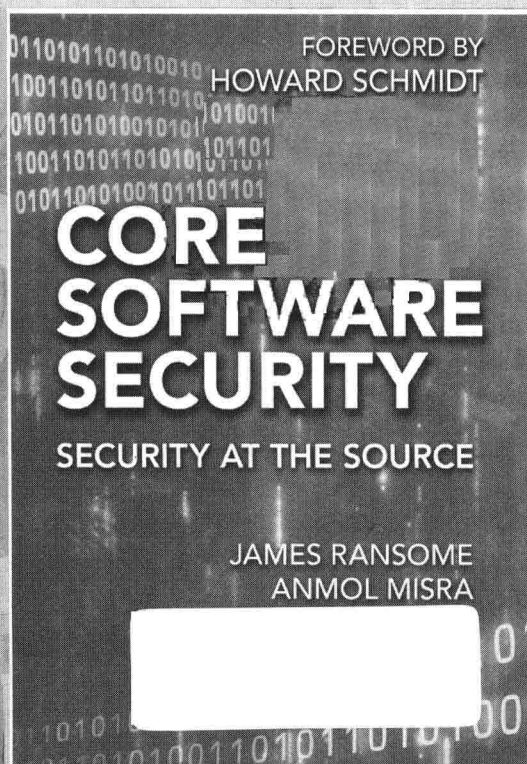
计 算 机 科 学 丛 书

软件安全

从源头开始

[美] 詹姆斯·兰萨姆 (James Ransome) 著
安莫尔·米斯拉 (Anmol Misra) 著
丁丽萍 卢国庆 李彦峰 穆海蓉 曹原野 译
宋宇宁 审校

Core Software Security
Security at the Source



机械工业出版社
China Machine Press

图书在版编目 (CIP) 数据

软件安全：从源头开始 / (美) 詹姆斯·兰萨姆 (James Ransome), (美) 安莫尔·米斯拉 (Anmol Misra) 著; 丁丽萍等译. —北京: 机械工业出版社, 2016.7

(计算机科学丛书)

书名原文: Core Software Security: Security at the Source

ISBN 978-7-111-54023-6

I. 软… II. ①詹… ②安… ③丁… III. 软件开发-安全技术 IV. TP311.52

中国版本图书馆 CIP 数据核字 (2016) 第 130357 号

本书版权登记号: 图字: 01-2016-0682

Core Software Security: Security at the Source by James Ransome, Anmol Misra (9781466560956).

Copyright © 2014 by Taylor & Francis Group, LLC.

Authorized translation from the English language edition published by CRC Press, part of Taylor & Francis Group LLC. All rights reserved.

China Machine Press is authorized to publish and distribute exclusively the Chinese (Simplified Characters) language edition. This edition is authorized for sale in the People's Republic of China only (excluding Hong Kong, Macao SAR and Taiwan). No part of this publication may be reproduced or distributed in any form or by any means, or stored in a database or retrieval system, without the prior written permission of the publisher.

Copies of this book sold without a Taylor & Francis sticker on the cover are unauthorized and illegal.

本书原版由 Taylor & Francis 出版集团旗下 CRC 出版公司出版, 并经授权翻译出版。版权所有, 侵权必究。

本书中文简体字翻译版授权由机械工业出版社独家出版并限在中国大陆地区销售。未经出版者书面许可, 不得以任何方式复制或抄袭本书的任何内容。

本书封面贴有 Taylor & Francis 公司防伪标签, 无标签者不得销售。

本书阐述什么是人类可控制管理的安全软件开发过程, 书中给出一种基于经验的方法, 来构建最好的安全软件开发模型, 以应对安全问题。本书分为三部分, 共 10 章。第 1 章简要介绍软件安全领域的主题及其重要性; 第 2 章讲解软件安全的难点以及 SDL 框架; 第 3 ~ 8 章揭示如何将 SDL 及其最佳实践映射到一个通用的 SDLC 框架; 第 9 章从资深软件安全架构师的角度给出关于成功方案的想法, 并且解读在开发安全软件时针对典型挑战的一些真实方法; 第 10 章结合现实世界中的安全威胁, 描述如何用合理的架构设计、实现与管理的 SDL 程序来提高安全性。

出版发行: 机械工业出版社 (北京市西城区百万庄大街 22 号 邮政编码: 100037)

责任编辑: 谢晓芳

责任校对: 董纪丽

印刷: 中国电影出版社印刷厂

版次: 2016 年 8 月第 1 版第 1 次印刷

开本: 185mm × 260mm 1/16

印张: 13

书号: ISBN 978-7-111-54023-6

定价: 69.00 元

凡购本书, 如有缺页、倒页、脱页, 由本社发行部调换

客服热线: (010) 88378991 88361066

投稿热线: (010) 88379604

购书热线: (010) 68326294 88379649 68995259

读者信箱: hzjsj@hzbook.com

版权所有·侵权必究

封底无防伪标均为盗版

本书法律顾问: 北京大成律师事务所 韩光 / 邹晓东

文艺复兴以来，源远流长的科学精神和逐步形成的学术规范，使西方国家在自然科学的各个领域取得了垄断性的优势；也正是这样的优势，使美国在信息技术发展的六十多年间名家辈出、独领风骚。在商业化的进程中，美国的产业界与教育界越来越紧密地结合，计算机学科中的许多泰山北斗同时身处科研和教学的最前线，由此而产生的经典科学著作，不仅肇划了研究的范畴，还揭示了学术的源变，既遵循学术规范，又自有学者个性，其价值并不会因年月的流逝而减退。

近年，在全球信息化大潮的推动下，我国的计算机产业发展迅猛，对专业人才的需求日益迫切。这对计算机教育界和出版界都既是机遇，也是挑战；而专业教材的建设在教育战略上显得举足轻重。在我国信息技术发展时间较短的现状下，美国等发达国家在其计算机科学发展的几十年间积淀和发展的经典教材仍有许多值得借鉴之处。因此，引进一批国外优秀计算机教材将对我国计算机教育事业的发展起到积极的推动作用，也是与世界接轨、建设真正的世界一流大学的必由之路。

机械工业出版社华章公司较早意识到“出版要为教育服务”。自1998年开始，我们就将工作重点放在了遴选、移译国外优秀教材上。经过多年的不懈努力，我们与 Pearson, McGraw-Hill, Elsevier, MIT, John Wiley & Sons, Cengage 等世界著名出版公司建立了良好的合作关系，从他们现有的数百种教材中甄选出 Andrew S. Tanenbaum, Bjarne Stroustrup, Brian W. Kernighan, Dennis Ritchie, Jim Gray, Alfred V. Aho, John E. Hopcroft, Jeffrey D. Ullman, Abraham Silberschatz, William Stallings, Donald E. Knuth, John L. Hennessy, Larry L. Peterson 等大师名家的一批经典作品，以“计算机科学丛书”为总称出版，供读者学习、研究及珍藏。大理石纹理的封面，也正体现了这套丛书的品位和格调。

“计算机科学丛书”的出版工作得到了国内外学者的鼎力相助，国内的专家不仅提供了中肯的选题指导，还不辞劳苦地担任了翻译和审校的工作；而原书的作者也相当关注其作品在中国的传播，有的还专门为其书的中译本作序。迄今，“计算机科学丛书”已经出版了近两百个品种，这些书籍在读者中树立了良好的口碑，并被许多高校采用为正式教材和参考书籍。其影印版“经典原版书库”作为姊妹篇也被越来越多实施双语教学的学校所采用。

权威的作者、经典的教材、一流的译者、严格的审校、精细的编辑，这些因素使我们的图书有了质量的保证。随着计算机科学与技术专业学科建设的不断完善和教材改革的逐渐深化，教育界对国外计算机教材的需求和应用都将步入一个新的阶段，我们的目标是尽善尽美，而反馈的意见正是我们达到这一终极目标的重要帮助。华章公司欢迎老师和读者对我们的工作提出建议或给予指正，我们的联系方式如下：

华章网站：www.hzbook.com

电子邮件：hzsj@hzbook.com

联系电话：(010) 88379604

联系地址：北京市西城区百万庄南街1号

邮政编码：100037



华章科技图书出版中心

译者序

Core Software Security: Security at the Source

一直以来，无论是系统软件还是应用软件，人们在其开发过程中都很少关注安全问题。总是在软件上线后，发现或者被发现了安全问题才意识到有需要解决的安全问题。于是，一个补丁接一个补丁地弥补。用一段时间以后，软件已经补丁摞补丁，而安全问题依然层出不穷。如果开发人员不了解软件安全的基本概念和问题，不了解安全设计的基本方法，不知道安全漏洞的类型有哪些，不能设计出针对安全性的测试用例，就写不出安全可靠的代码，软件的安全就无从谈起。为此，本书作者提出了软件核心安全的根本在于把安全贯穿于软件开发过程的始终，对于安全开发生命周期（Security Development Lifecycle）的概念和模型进行了详细的分析阐述。

本书旨在让软件开发人员把安全理念贯穿于软件开发过程中，对软件的安全需求进行认真分析，把安全融入软件的概要设计和详细设计中，从而写出具有安全防护功能的代码，设计包含针对常见安全漏洞的测试用例，构建安全的运行环境。最终，从源头上关注软件安全，提高软件自身的安全性，减少安全漏洞带来的损失，降低软件安全开发的成本。

由于时间仓促，本书的译文难免有错误和不足之处，恳请广大读者批评指正。

丁丽萍

2016年7月9日于北京

全球网络安全威胁正在持续地（即便不是每天）增加。一个一直困扰着我们的问题是如何应对当前的全球网络安全威胁。作者通过本书回答了这个问题，即要开发安全漏洞尽可能少的软件。换言之，我们应该着眼于源头安全，而不是仅仅采取如试图保护网络基础设施等阻挡入侵的方法来解决安全问题。边界安全和深度防御在安全领域中占有一席之地，但软件自身的安全是安全防护的第一关，应该是第一位的。即使在软件源头中存在较少的漏洞，这些漏洞也足以被利用，成为侵犯民族和国家经济利益的武器，或者成为有组织犯罪的网络武器储备。因此，我们不仅要把软件做得更好、更安全，同时，根据现实世界的经验，必须保证该解决方案具有较好的成本效益、操作相关性和可行性以及投资的可行性。源头安全需要软件安全，这是网络基础设施安全的核心。20 年以来，我们一直不可否认的一个事实是软件已成为我们的关键基础设施和日常生活方方面面的重要组成部分。我们注意到软件已经融入我们日常生活使用的各种各样的物品中——从家庭智能电表到汽车都含有软件。遗憾的是，软件安全并没有像软件本身那样以相同的速度发展。许多软件产品依然在这样一种环境下开发：它们在发布后解决问题而不是保证第一次做的就是正确的。这里，有两个主要的问题。

1. 第一个问题是软件漏洞普遍存在。正在寻找和利用软件漏洞的那些人有着充足的漏洞可挖掘，因此，我们必须确保漏洞管理足够好，而且，还必须将目光投向未来并扪心自问：“怎样做才能让我们的生活越来越依赖的后续软件中不会存在这种类型的漏洞？”这个问题的答案显得尤为重要。因为对于企业来说，减少这些漏洞以及在软件开发过程中阻止它们的出现是非常有益的。使用 SDL 构建软件安全比软件发布以后的系统恢复和漏洞修复的成本更低。

2. 另一个问题是我们需要开始关注称为“0-day 漏洞”的整整一代漏洞。坚持可靠 SDL 的最佳实践，通过从一开始就不允许漏洞发生的方式，找到可能消除 0-day 漏洞的方法，将节省企业的资金投入，使得软件及其用户更安全，关键基础设施更具弹性。总之，这些将对我们所有人都更有益。

作为卓越代码软件保障论坛（Software Assurance Forum for Excellence in Code, SAFECode）的执行董事，我现在重点关注开发人员的安全培训。SAFECode 是一个非营利性的组织，专门致力于通过先进有效的软件保证方法，增加信息、通信技术产品和服务的可信性。软件工程师人员安全意识和教育的匮乏，可能是努力实现软件安全计划组织的显著障碍。然而，给予软件开发人员更好的培训，使他们具备编写安全代码所需的技能，仅仅是软件安全方程式的变量之一。软件项目受限于成本和严格的时间表。在这种情况下，因为某处走捷径而牺牲安全是不可避免的。成本、时间和资源通常是软件开发支持安全的三要素，如果你牺牲三者中的任意一个，安全和质量都会深受其害。软件开发环境是围绕程序员建立的，他们受到各个方面的压力：更快地工作，走捷径，以安全和质量为代价编写更多的代码。

100% 安全的软件和系统是不存在的，但开发者和他们的管理者应始终力求最大限度地缓解风险，目的是使攻击者以未经授权的方式进行访问变得更难：

- 不得已利用可追溯的和明显的入侵手段访问，以致他们被立刻发现；

- 花费很多时间试图访问系统，以致最终被注意到；
- 放弃并转向更容易的攻击目标。

确保每一个接触到产品开发生命周期的人拥有支持组织的软件安全过程所需的知识，是致力于软件安全成功的任何组织的一个基本挑战。目标是消除组织在开发传统程序过程中面临的受限于资源约束和知识隔绝的痛苦。

开发者往往需要在巨大的压力下和预算内按时提供更多的功能。很少一部分开发者能够抽出时间来审查他们的代码以发现潜在的安全漏洞。当他们真的抽出时间的时候，他们往往没有经过安全编码方面的培训，缺乏自动化工具、嵌入式流程和程序、资源等，以防止黑客使用数百个常见的漏洞利用技术来触发恶意攻击。不管你喜欢与否，开发者辛勤工作的劳动果实往往是恶意黑客攻击的受害者。那么软件厂商能够做些什么呢？答案的很大一部分是比较老套的：开发者需要获得激励、更好的工具和适当的培训。

遗憾的是，当前的商业意识倾向于不生产安全的软件产品。任何解决方案都需要将其当作一个根本的市场失败问题，而不是简单地希望它不是真的。如果安全性是一个企业追求的目标，那么它需要具有商业意义。最后，安全性要求实际上与任何商业目标都是相同的，应该被视为同等重要。雇主应该期望他们的雇员对他们的工作感到自豪并拥有一定的责任感。雇员应该期望他们的雇主提供完成工作所需的工具和培训。有了这些期望的建立和目标的达成，也许软件行业可以通过减少软件漏洞，更好地提高产品的安全性。

本书讨论了一种过程方法，该方法认为安全性在软件开发过程中发挥着至关重要的作用。本书面向企业高管、各个级别的管理和技术领导者，对他们而言，本书所述的这种方法是独特的。本书同样面向学生和开发者，他们可以了解软件开发过程，树立安全意识，并找到资源帮助他们实现本书所介绍的方法。书中的信息给管理人员、项目经理和技术领导者提供了一定的基础，以改善他们创建的软件、提高应用软件的安全性和质量，并保护隐私信息。

软件开发者知道如何编写软件，以提供高水平的安全性和鲁棒性。那么，为什么软件开发者不练习这些技巧呢？本书将分两部分来回答这个问题。

1. 软件是否安全取决于人们对于程序的分析结果，这些分析包括软件如何使用、在什么条件下使用和将要部署的环境中它必须满足的安全要求是什么。SDL 还必须延伸到产品发布之外，因为如果计划外的操作环境中软件背后的假设和它们此前隐含的需求不成立，该软件可能不再是安全的。如果需要重新设计一个完整的产品，SDL 过程可能部分或全部重新开始。从这个意义上说，作者建立了所需的准确和有意义的的目标、管理的度量并为开发软件确立了一个实例。本书还假定不是所有的安全要求均优先于开发过程，并描述它们衍生、分析和验证的过程。

2. 软件高管、领导者和管理人员必须支持鲁棒的编码实践，以及一个业务相关的 SDL 所需的安全性增强，同时支持这种类型工作所需的人员需求、调度、预算和资源分配。作者出色地为这些管理者描述了过程、需求和度量管理，这样他们能够准确地评估一个 SDL 所需的影响和资源，使得他们在组织和环境中的工作能够相对最佳。鉴于这是一个依据真实生活、实地挑战和经验设计的方法，作者描述了如何思考问题，以制定有效的方法并遵照业务流程进行管理。

我特别喜欢第 9 章的作者 Brook Schoenfield 给本书补充的内容，带给我们一个针对企业和软件安全架构师的成熟结论。这个结论借助“第一线”的丰富经验，讲解在“现实世界”

中安全架构如何适应 SDL 过程。特别地，他提供了一个独特和重要的办法来解决 SDL 与安全架构方面的需求，而且这个方法已经过实地测试并真正使用在敏捷软件开发过程中。

我已经认识 James Ransome 博士多年，很高兴他选择信息安全作为他第 10 本书的主题。除了在政府和企业领域担任一些网络安全高级领导职务外，我最近刚刚就任总统特别助理和联邦政府网络安全协调员。我可以自信地说，这是目前信息和全球网络安全最关键的领域。业务和流程问题一直是并将继续是比技术更重要的问题。本书为目前典型的培训资源添砖加瓦，因为它使用了众所周知的 SDL 并提供了运营、业务和成本效益指标。我相信，James Ransome 和 Anmol Misra 所写的书已经一语道破这一主题，将作为专业人士的实践指南和学术教材服务社会若干年。

Hon. Howard A. Schmidt

Ridge Schmidt Cyber 合伙人

SAFECode 执行董事

Hon. Howard A. Schmidt 简介

Hon. Howard A. Schmidt 是战略咨询公司 Ridge Schmidt Cyber 的合伙人。该公司是一个行政服务公司，帮助企业和政府的领导者引领网络安全不断增长的需求。Tom Ridge 也担任该职务，他是美国国土安全部的首位部长。Schmidt 教授还担任 SAFECode 的执行董事。

Schmidt 教授通过长达 40 年杰出的职业生涯，将企业人才、国防、情报、执法、隐私、学术界和国际关系融合在一起。最近他就任总统特别助理和联邦政府网络安全协调员。在此期间，Schmidt 协调各部门网络安全政策的制定和实施，并与联邦、州、地方、国际和私营部门的网络安全合作伙伴通力合作。

此前，Schmidt 教授担任信息安全论坛（Information Security Forum, ISF）的总裁兼 CEO。在 ISF 任职之前，他担任 eBay 公司的副总裁、首席信息安全官和首席安全战略官。此前，他担任微软公司的首席安全官。他还担任过美国国土安全部 US-CERT 合作伙伴计划的首席安全战略官。Schmidt 还拥有超过 26 年的军旅生涯。开始是在空军服役，后来加入了美国亚利桑那州空军国民警卫队。在空军服役期间，他曾担任许多军事和民间职务，最高职务为特别调查办公室的监事会特别代理（AFOSI）。他在刑事调查部门的计算机罪案组担任陆军预备役特别代理长达 12 年，在亚利桑那州钱德勒警察署担任警察。

Schmidt 教授于凤凰城大学获得工商管理学士学位（bachelor's degree in business administration, BSBA）和组织管理硕士学位（master's degree in organizational management, MAOM）。他还拥有人文学名誉博士学位。Schmidt 是爱达荷州立大学的讲座教授，兼卡内基梅隆大学 CyLab 实验室的特聘研究员和波耐蒙隐私研究所（Ponemon Privacy Institute）的特聘研究员。

Schmidt 还是一名业余无线电操作员（W7HAS）、私人飞行员、户外运动者和狂热的哈雷 - 戴维森（Harley-Davidson）骑手。他夫人 Raemarie J. Schmidt 是一个退休的取证科学家和研究人员，是计算机取证领域的讲师。总之，他们是自豪的父母和快乐的祖父母。

软件驱动机器的时代在过去的几年里取得了飞跃式的发展。战斗机飞行员、股票交易所场内交易员、医生、工业生产者和发电厂运营商等专业人员的工作对武器系统、医疗系统以及国家基础设施关键要素的运行至关重要，而这些行业已经或正在迅速地被软件接管。这是革命性的一步。从前必须要人类大脑完成的复杂的、非重复性的任务，现在已经被软件驱动程序控制的机器大脑和神经系统所取代。这种变化体现在政府、军队、罪犯、活动家和其他对手可以尝试破坏、修改或影响国家基础设施的改变，也体现在社会和文化方式的转变。同样，这种变革对企业也是有一定影响的，比如，这些年来我们看到的越来越多的网络商业间谍案件。以前的大规模军队、昂贵和毁灭性的武器系统与平台、武装抢劫、信息窃取、暴力抗议活动和武装叛乱很快被所谓网络战、网络犯罪和网络行为所替代。

这些网络方法可能最终就像之前使用的技术一样产生深远的影响，而潜在的软件漏洞如果被利用，则可能会导致以下问题。

- 全部或部分基础设施被中断，包括电网、核电站、通信介质和应急系统。
- 化工厂被用于产生大规模的爆炸和剧毒云。
- 远程控制、修改或破坏关键的武器系统或平台。
- 导致监控系统被篡改或不可用。
- 经济剥削或讹诈犯罪。
- 操控金融市场和投资。
- 通过改变医疗支持系统或设备、手术计划或药物处方来实施谋杀或伤害人体健康。
- 通过网站毁损或底层 Web 应用的关闭和破坏修改自动投票软件、进行敲诈勒索或导致品牌信用降级，从而产生政治暴动或者导致一些人群的特殊利益受到影响。

网络方法的一个副作用就是提供给我们以之前想都不敢想的大规模、远程和匿名方式解决上述问题的能力。这一能力是在受到司法保护的位置通过远程探测和入侵的方式解决上述问题。同时，这也赋予了政府、犯罪集团和活动家特殊的能力，包括代理主要犯罪人用于逃避责任、避免接受检测和规避政治后果。

虽然有很多关于网络安全的宣传，但真正的致命弱点是（不安全的）软件。这些软件提供总体控制或变更如上所述的一个目标的潜在能力。软件安全的关键性在于，我们迅速步入了新的时代，从前被忽视的关于人类思维被软件驱动机器所替代的这件事也不容小觑。正是出于这个原因，我们撰写了这本书。与此相反，在可预见的将来，软件程序将继续由人类编写。这也意味着，新的软件将继续构建在遗留代码或软件之上，而这些遗留代码在编写时安全还不被重视或者是在复杂攻击盛行之前编写的。只要人类写程序，成功的关键就是保障有关计划的实施，使得软件开发项目的过程更加高效和有效。尽管本书的方法包括软件安全人员、软件安全流程和软件安全技术方法，但是我们认为在软件安全中人是重要的组成部分，原因是软件的开发、管理、使用都是由人来完成的。以下是软件安全性的逐步实施过程，该过程与当今的技术、运营、业务和开发环境相关，并且强调人可以做什么去控制和管理流程的最佳实践与度量方式。我们将永远面临安全问题，但是本书有助于在软件最终发布或部署时

尽量减少安全问题。希望你能喜欢我们的书，就像我们喜欢写它一样。

本书内容

本书讲述关乎当今科技、运作、商业及开发环境的软件安全过程。作者着重讲述什么是人类可控制、管理的软件开发过程，并用一种较好的实践方式和度量方式来表达。虽然安全问题将会始终存在，但是本书将教你如何将公司的能力最大化，同时在软件发布前将软件被攻击的概率最小化，以及如何将安全构建融入开发过程之中。作者拥有世界五百强公司的工作经历，目睹过一定数量的软件安全开发生命周期（SDL）故障。本书采用一种基于经验的方法，来构建最好用的 SDL 模型，以应对前面提到的问题，并在 SDL 安全软件开发模型中得到解决。本书开篇概要介绍 SDL，然后讲解将 SDL 实践方法映射到 SDL 模型，同时解释如何运用此模型来建立并管理成熟的 SDL 程序。虽然安全并不是近些年业界编写软件过程中的天然组成部分，但是，作者相信，将安全开发加入到软件开发中将是可能的、可行的，也是切实必要的。作者同时认为，本书中提到的软件安全实例以及模型将给所有读者留下清晰的印象，无论是经理还是主管或是从业人员都将在其中获益。

读者对象

本书面向任何对企业级软件安全感兴趣的人，包括产品安全和质量主管、软件安全架构师、安全顾问、软件开发工程师、企业 SDLC 项目经理、首席信息安全官、首席技术官和首席隐私官。如果你了解企业级软件开发过程中是如何实现软件安全的，本书将是一本不可错过的好书。

本书结构

本书分为三部分，共 10 章。第 1 章简要介绍软件安全领域的主题及其重要性。第 2 章讲述软件安全的难点以及 SDL 框架。第 3 ~ 8 章讲述如何将 SDL 及其最佳实践映射到一个通用的 SDLC 框架。根据第 3 ~ 8 章描述的解决方案，第 9 章叙述一个老练的软件安全架构师关于成功方案的看法，并解读在开发安全软件时针对典型挑战的一些真实方法。第 10 章结合现实世界中的安全威胁，描述我们如何用合理的架构设计、实现与管理的 SDL 程序来提高安全性。

假设

本书假设读者了解基本的软件开发过程（方法论）以及基本的安全概念。建议对 SDL、不同种类的安全测试，以及安全架构有所了解，但是不作要求。对于大部分主题，在深入探讨具体问题前，我们会循序渐进地介绍相关知识。

致谢

写书就像旅行，如果没有导师、朋友、同事还有家人的支持，它将无比困难。在写作过程中很多人给予了我帮助。首先，我们要感谢 CRC 出版社 John Wyzalek 编辑的耐心帮助、支持以及贡献。我们还要感谢 DerryField Publishing 产品组的 Theron Shreve、Lynne Lackenbach 和 Marje Pollack。

我们都要感谢 Hon. Howard A. Schmidt (Ridge Schmidt Cyber 合伙人; SAFECode 执行董事; 联邦政府总统及网络安全司前特别助理), 并感谢 Dena Haritos Tsamitis (网络信息研究所主管; 卡耐基梅隆大学 CyLab 实验室教育、训练、外展主管) 对项目的支持。我们同样要感谢 Brook Schoenfield 对项目的帮助。感谢他加入团队并证明有另一种不同于现有方法的软件安全架构、部署和管理方法, 同时感谢他撰写了本书第 9 章。我们要感谢整个网络安全社区, 我们属于其中的一员并引以为荣。最后我们衷心感谢这些年所有共事并合作过的人。

——James Ransome 和 Anmol Mistra

借此机会, 我真心感谢我的妻子 Gail。感谢她的耐心和理解, 也感谢她在初步校对中的工作。特别要感谢的是我的合作者 Anmol Misra。他作为合作者加入我们关键消息的开发已经三年多了, 并且一并完成了这本书。另一位要特殊感谢的是 Howard Schmidt。感谢他为本书作序。我们彼此分享做实干者的热情。最后和各位读者分享 Walter Bagehot 的一句话: “生命里最快乐的事就是做别人声称做不成的事。”

——James Ransome

多年来许多人指导并帮助过我。我想感谢所有这些人, 可惜篇幅有限。你知道我说的是谁, 我想感谢他们的耐心、鼓励与支持。我要感谢我的合作者 James Ransome。多年来他一直是我的良师益友。他对我的帮助无法言表。最后我想借此机会感谢我的家人: 妈妈、爸爸、Sekhar、Anupam 和 Mausi。没有他们无私的支持和爱护, 我是不可能完成任何事情。他们曾经问我在写完这本书后是否能休假, 是否能有一个“正常”的日程安排。我想说, 我会的, 现在就会。希望如此。

——Anmol Misra

James Ransome

James Ransome 博士是负责产品安全的高级总监，全面负责 McAfee 产品安全项目，该项目是一个企业范围的举措，它支持 McAfee 业务部门给客户提供一流的、安全的软件产品。在担任该职位期间，James 负责设置安全战略，管理 McAfee 业务单元相关的安全活动，保持与 McAfee 产品工程师的联系并与其他领导者合作，以帮助定义和构建产品的安全功能。

他职业生涯的特点是在私人和公共行业担任领导职务，其中包括三个首席信息安全官（Chief Information Security Officer, CISO）和四个首席安全官（Chief Security Officer, CSO）的职务。在进入企业领域前，James 有 23 年从政经历，包括美国情报界、联邦执法部门和国防部的各种职务。

James 拥有信息系统博士学位。他的博士论文包括开发 / 测试安全模型、体系结构，并提供了配合有线 / 无线网络安全方面领先的实践，其论文是信息安全保障教育程序 NSA/DHS 中心的卓越学术成果的一部分。他是多本信息安全书籍的作者，本书是第 10 本。James 是计算与信息学科国际荣誉协会 Upsilon Pi Epsilon 的成员。他是一名注册信息安全经理（Certified Information Security Manager, CISM），一名信息系统安全认证专家（Certified Information Systems Security Professional, CISSP），还是波耐蒙研究所（Ponemon Institute）的特聘研究员。

Anmol Misra

Anmol Misra 是信息安全领域的一位作家和拥有丰富经验的安全专业人士。他的专长包括移动和应用安全、漏洞管理、应用和基础设施的安全评估，以及安全代码审校。他在思科公司信息安全组担任项目经理。在此期间，他主要负责制定和实施安全策略与计划，以把安全最佳实践纳入思科主导的产品的各个方面。就职于思科公司之前，Anmol 是安永国际会计公司的高级顾问。在这个岗位上，他给财富 500 强客户提供定义和改进信息安全计划与实践的咨询服务。他帮助企业降低 IT 安全风险，并通过改善其安全状况使其遵从法规。

Anmol 是《Android Security: Attacks and Defenses》的合作者，是《Defending the Cloud: Waging War in Cyberspace》的贡献作者。他于卡内基梅隆大学获得信息网络硕士学位和计算机工程工学学士学位。他工作于加利福尼亚州的旧金山。

目 录

Core Software Security: Security at the Source

出版者的话
序
前言
作者简介

第 1 章 引论..... 1

- 1.1 软件安全的重要性和相关性..... 1
- 1.2 软件安全和软件开发生命周期..... 4
- 1.3 代码的质量与安全..... 6
- 1.4 SDL 三个最重要的安全目标..... 6
- 1.5 威胁建模和攻击面验证..... 7
- 1.6 本章小结：期望从本书中学到什么... 8
- 参考文献..... 8

第 2 章 安全开发生命周期..... 11

- 2.1 克服软件安全中的挑战..... 11
- 2.2 软件安全成熟度模型..... 12
- 2.3 ISO/IEC 27034：信息技术、
安全技术、应用安全..... 13
- 2.4 其他 SDL 最佳实践的资源..... 14
 - 2.4.1 SAFECODE..... 14
 - 2.4.2 美国国土安全软件保障计划... 14
 - 2.4.3 美国国家标准与技术研究院... 15
 - 2.4.4 MITRE 公司公共计算机漏洞
和暴露..... 16
 - 2.4.5 SANS 研究所高级网络安全
风险..... 17
 - 2.4.6 美国国防部网络安全与信息
系统信息分析中心..... 17
 - 2.4.7 CERT、Bugtraq 和
SecurityFocus..... 17
- 2.5 关键工具和人才..... 17
 - 2.5.1 工具..... 18
 - 2.5.2 人才..... 19
- 2.6 最小特权原则..... 21

- 2.7 隐私..... 22
- 2.8 度量标准的重要性..... 22
- 2.9 把 SDL 映射到软件开发生命周期... 24
- 2.10 软件开发方法..... 28
 - 2.10.1 瀑布开发..... 28
 - 2.10.2 敏捷开发..... 29
- 2.11 本章小结..... 31
- 参考文献..... 31

第 3 章 安全评估 (A1)： SDL 活动与最佳实践..... 35

- 3.1 软件安全团队提早参与项目..... 35
- 3.2 软件安全团队主持发现会议..... 37
- 3.3 软件安全团队创建 SDL 项目计划... 37
- 3.4 隐私影响评估计划启动..... 38
- 3.5 安全评估 (A1) 成功的关键因素
和度量标准..... 41
 - 3.5.1 成功的关键因素..... 41
 - 3.5.2 可交付成果..... 43
 - 3.5.3 度量标准..... 44
- 3.6 本章小结..... 44
- 参考文献..... 44

第 4 章 架构 (A2)： SDL 活动与最佳实践..... 46

- 4.1 A2 策略一致性分析..... 46
- 4.2 SDL 策略评估和范围界定..... 48
- 4.3 威胁建模 / 架构安全性分析..... 48
 - 4.3.1 威胁建模..... 48
 - 4.3.2 数据流图..... 50
 - 4.3.3 架构威胁分析和威胁评级..... 53
 - 4.3.4 风险缓解..... 65
- 4.4 开源选择..... 68
- 4.5 隐私信息收集和分析..... 69
- 4.6 成功的关键因素和度量标准..... 69

4.6.1 成功的关键因素	69	7.3 渗透测试	114
4.6.2 可交付成果	70	7.4 开源许可审查	116
4.6.3 度量标准	70	7.5 最终安全性审查	117
4.7 本章小结	71	7.6 最终隐私性审查	119
参考文献	71	7.7 成功的关键因素	120
第 5 章 设计和开发 (A3):		7.8 可交付成果	121
SDL 活动与最佳实践	74	7.9 度量标准	122
5.1 A3 策略一致性分析	74	7.10 本章小结	122
5.2 安全测试计划构成	74	参考文献	124
5.3 威胁模型更新	81	第 8 章 发布后支持 (PRSA1 ~ 5) ...	125
5.4 设计安全性分析和检查	81	8.1 合理调整软件安全组	125
5.5 隐私实现评估	83	8.1.1 正确的组织定位	125
5.6 成功的关键因素和度量标准	85	8.1.2 正确的人	127
5.6.1 成功的关键因素	85	8.1.3 正确的过程	127
5.6.2 可交付成果	86	8.2 PRSA1: 外部漏洞披露响应	130
5.6.3 度量标准	87	8.2.1 发布后的 PSIRT 响应	130
5.7 本章小结	88	8.2.2 发布后的隐私响应	133
参考文献	88	8.2.3 优化发布后的第三方响应	133
第 6 章 设计和开发 (A4):		8.3 PRSA2: 第三方审查	134
SDL 活动与最佳实践	90	8.4 PRSA3: 发布后认证	135
6.1 A4 策略一致性分析	90	8.5 PRSA4: 新产品组合或云部署	
6.2 安全测试用例执行	92	的内部审查	135
6.3 SDLC/SDL 过程中的代码审查	94	8.6 PRSA5: 安全架构审查和基于	
6.4 安全分析工具	97	工具评估当前、遗留以及并购	
6.4.1 静态分析	99	的产品和解决方案	136
6.4.2 动态分析	101	8.6.1 遗留代码	136
6.4.3 模糊测试	103	8.6.2 兼并和收购	137
6.4.4 人工代码审查	104	8.7 成功的关键因素	138
6.5 成功的关键因素	106	8.8 可交付成果	139
6.6 可交付成果	107	8.9 度量标准	140
6.7 度量标准	107	8.10 本章小结	140
6.8 本章小结	108	参考文献	140
参考文献	108	第 9 章 将 SDL 框架应用到现实	
第 7 章 发布 (A5):		世界中	142
SDL 活动与最佳实践	111	9.1 引言	142
7.1 A5 策略一致性分析	111	9.2 安全地构建软件	145
7.2 漏洞扫描	113	9.2.1 编写安全的代码	146
		9.2.2 人工代码审查	149

- 9.2.3 静态分析..... 150
- 9.3 确定每个项目的正确行为..... 153
- 9.4 架构和设计..... 161
- 9.5 测试..... 167
 - 9.5.1 功能测试..... 168
 - 9.5.2 动态测试..... 168
 - 9.5.3 攻击和渗透测试..... 171
 - 9.5.4 独立测试..... 172
- 9.6 敏捷：冲刺..... 172
- 9.7 成功的关键因素和度量标准..... 175
 - 9.7.1 安全编码培训计划..... 175
 - 9.7.2 安全编码框架 (API)..... 175
 - 9.7.3 人工代码审查..... 176
 - 9.7.4 独立代码审查和测试
(专家或第三方)..... 176
 - 9.7.5 静态分析..... 176
 - 9.7.6 风险评估法..... 176
 - 9.7.7 SDL 和 SDLC 的集成..... 176
 - 9.7.8 架构人才的发展..... 176
- 9.8 度量标准..... 177
- 9.9 本章小结..... 177
- 参考文献..... 178

- 第 10 章 集成：应用 SDL 防止现实的威胁..... 180
 - 10.1 战略、战术和特定于用户的软件攻击..... 180
 - 10.1.1 战略攻击..... 181
 - 10.1.2 战术攻击..... 182
 - 10.1.3 特定于用户的攻击..... 182
 - 10.2 应用适当设计、管理和集中的 SDL 克服组织与业务挑战..... 182
 - 10.3 软件安全组织的现状和影响力... 183
 - 10.4 通过合理的政府管理克服 SDL 审计和法规挑战..... 183
 - 10.5 软件安全的未来预测..... 184
 - 10.5.1 坏消息..... 184
 - 10.5.2 好消息..... 185
 - 10.6 总结..... 185
 - 参考文献..... 186

- 附录 关键的成功因素、可交付成果、SDL 模型每个阶段的指标..... 189

引 论

欢迎阅读此书，本书旨在关注未来信息安全领域最重要的话题：软件安全（software security）。下面几节将涵盖五大主题，突出说明软件安全的需求、价值和挑战。这将为本书的剩余部分定下基调。我们将描述一个软件安全模型：应用一个操作上相关且可控的安全开发生命周期（Security Development Lifecycle, SDL）来构建安全软件。SDL 适用于所有的软件开发生命周期（Software Development Lifecycle, SDLS）。五大主题以及在第 1 章引入它们的原因如下。

1. **软件安全的重要性和相关性。**软件是我们在现实世界中做任何事情的关键，同时，软件也分布在最关键的系统中。基于此，软件的安全设计是至关重要的。大多数信息技术（Information Technology, IT）相关的安全解决方案已经能够有效地降低不安全软件带来的风险。为了证明一个软件安全程序的合理性，必须知晓没有构建安全软件带来的金钱成本和其他风险的重要性与相关性，以及构建安全软件的重要性、相关性和成本。总而言之，软件安全同样是一个商业决定，因为它关注避免安全风险。

2. **软件安全和软件开发生命周期。**在这里，重要的一点是要区分在软件开发中我们熟悉的软件安全（software security）和应用程序安全（application security）。尽管这两个术语经常互用，但是我们仍然需要区分它们。因为实现这两个目的的程序在管理过程中存在明显的不同。在模型中，软件安全表示在 SDLC 中，应用 SDL 构建安全的软件，而应用程序安全表示发布后运行过程中软件和系统的保护。

3. **高质量和安全代码。**尽管安全代码未必是高质量代码，同时高质量代码也未必是安全代码，但是软件的开发过程是基于高质量和安全代码原则的。你不能拥有不安全的高质量代码，更不能拥有劣质的安全代码，它们相辅相成。至少，质量和软件安全程序应该在开发过程中紧密结合；理想情况下，它们应该是同一组织的组成部分，以及软件开发工程部门的一部分。这将在本书中后面的章节中从组织和操作角度进一步讨论。

4. **三个最重要的 SDL 安全目标。**所有软件安全分析和构建的核心是三个最重要的安全因素：保密性（Confidentiality）、完整性（Integrity）和可用性（Availability），也称为 C.I.A. 模型。为了确保软件开发是安全的，上述三个特性必须一直作为整个 SDL 过程中的主要组成部分。

5. **威胁建模和攻击界面验证。**威胁建模和攻击界面验证是 SDL 中最耗时和难以理解的部分。在当今的敏捷软件开发中，必须正确处理这个问题，否则无法保证软件安全。SDL 中的威胁建模和攻击界面验证，将最大限度地避免软件产品发布后发现安全漏洞。我们认为这个功能非常重要，因此本书将专门用一节和一章来关注这个主题。

1.1 软件安全的重要性和相关性

2005 年美国总统信息技术顾问委员会（PITAC）报告指出：“通用的软件工程实践蕴含着危险的错误，比如处理不当的缓冲区溢出，这导致每年存在成百上千的攻击程序危害成千上

万的计算机。”^[1]发生这种情况的主要原因是“当今商业软件工程缺乏在合理成本内构建高质量、安全产品的科学基础和严格管理。”^[2]

高德纳咨询公司（Gartner Group）报道超过 70% 的通用商业安全漏洞出现在软件应用中，而不是网络边界内^[3]。因此关注应用安全，旨在降低糟糕的软件开发、集成和部署带来的风险。结果是，软件保障迅速成为金融、政府、制造业等部门信息保障（Information Assurance, IA）关注的领域，并且用于降低不安全代码带来的风险：将安全构建到软件开发生命周期中是一种良好的商业意识。

美国国土安全部 2006 年草案“软件生命周期中的安全”描述如下：

安全的软件和不安全的软件最关键的区别在于描述、设计和开发软件等过程与实践中的性质……主要采用安全增强的过程和实践，尽可能早地修复软件开发生命周期中潜在的安全漏洞，比当前普遍采用的频繁开发和发布运行中软件补丁的方法更加经济实用。^[4]

在 2011 年美国 RSA 会议上，大家着重强调了云安全问题，但几乎没有讨论如何解决该问题；然而，在 2012 年 RSA 会议上，几乎全是关于解决一年前确定的云安全问题的研究。相同的事情也发生在 2012 年，开始于一些重要的学术会议，随后在 2013 年关于软件安全解决方案的讨论继续成为一个主要关注点。例如，在 2012 年年初，《信息周刊》（Information Week）提出“外部代码审查”是 2012 年十大安全趋势之一^[5]，并且指出“业务职责是明确的：开发者必须花时间清楚地编写代码，并在代码投入生产之前根除任何可能的安全缺陷。”同样 2012 年 3 月 1 日也有一篇发表的热门安全文章，题目为“To Get Help with Secure Software Development Issues, Find Your Own Flaws”，它成为 2012 年旧金山举办的 RSA 专题讨论会上最精彩的部分。^[6]这次专题讨论会做了大量的工作以识别一些关键问题，但并没有解决识别出的软件安全挑战。然而，事情开始在 2012 年年中有所改变：2012 年 5 月召开的微软就业安全发展会议议程^[7]，不是关于微软的，更多的是带来安全软件开发思想的领导力标准，并分为三个独立的模块，包括“安全工程”“安全开发生命周期（SDL）& 商业”和“过程管理”来讨论工业和安全软件开发中最重要的安全问题的解决方案。这种趋势在 2012 年美国黑帽会议^[8]、2013 年 RSA 会议^[9]和 2013 年微软安全开发会议^[10]上得到延续。

想想看：是什么真正引发了当今世界大多数的信息安全问题？黑客和网络犯罪分子的主要目标又是什么？那就是不安全的代码。是什么已经迅速成为软件开发中投入最高的不必要的成本？那就是已投入市场的软件产品中不安全代码带来的缺陷。当这些缺陷被发现和（或者）被利用后，它们会导致当今产品开发周期中断以修复一些本应该在包含缺陷的产品开发过程中修复的错误；它们会导致产品发布日期推迟，因为参与当前项目的个人或者团队脱离该周期去修复以前发布产品的问题；同时它们会导致脆弱性范围渐变，因为在某一产品中发现的脆弱性可能影响网络、软件即服务（Software as a Service, SaaS）和云应用中其他产品的安全性。它们也会产生法律问题，如声誉下降和诸如在过去的几年里索尼（Sony）、赛门铁克（Symantec）和 RSA 经历的公共关系的噩梦。它们同样能够导致公司重大的责任。在有广泛的法规监管用户隐私和数据泄露的时代，企业应该承担的责任将迅速增加，即使对于大公司也一样。需要指出的是，即便是在高科技领域，消费者、客户、监管部门和媒体都已经开始意识到不仅要迫切解决软件安全问题，实际上更需要以一种结构化的安全软件开发生命周期或

者框架的方式解决这些问题。这样所有的注意力都将集中在那些开发软件代码的开发者身上，尤其是开发关键和敏感应用代码的程序员。检查这些程序员在他们自己的环境中是否坚持这样的实践，无论是采用传统 / 瀑布、Scrum/ 敏捷还是一种混合的开发方法。

全球经济的每一个部门，从能源到运输、财政、银行、电信、公共健康、应急服务、水、化学制品、防御、工业、事物、农业，以及邮政和运输部门，都依赖于计算机软件。实际上，软件的任何威胁都会对我们的生活构成威胁。因为全部潜在的危害都发生在利用编码缺陷的时候，因此产品不仅仅要保证正常的运行（质量），还必须是安全的（安全）。因此，我们坚信本节的主题与内容能够解决未来几年信息和网络安全将要面对的大多数严峻挑战。

许多人认为，当产品已开发并投入使用后，你依然可以修复软件漏洞。但这不是那么容易，因为修复漏洞的成本增加超过 SDLC，如图 1-1 所示，大多数与安全相关的工作是在发布以后（post-release）完成的，包括代码审核、补救、bug 修复、打补丁，也包括黑客防范措施。在云环境中，可能有多个版本的应用程序在运行，解决跨越所有应用程序的安全漏洞往往是一个挑战。云环境中某一版本应用程序的暴露可导致所有应用程序的漏洞被利用，除非制定了严格的网络隔离控制。就连这种隔离控制也都有可能被证明在复杂的攻击事件面前是脆弱的。

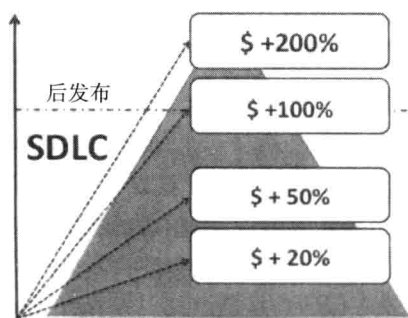


图 1-1 解决映射到 SDLC 阶段软件问题的成本

随着项目的生命周期逐渐成熟，解决软件问题相关的成本也在增加（见图 1-1）。1988 年，Barry Boehm 表示，在现场发现缺陷将花费 50 ~ 200 倍于较早纠正该缺陷所花费的费用^[11]。若干年以后，Barry Boehm 表示，对于小型非关键系统来说这个比例是 5 : 1。^[12]2008 年 Fortify 展示的数据表明，在需求分析层面校正安全缺陷的成本比现场修复这些软件安全缺陷的成本低 100 倍。^[13]我们都十分明确的一点是，无论使用什么数据，通过上述示例以及工业中使用的其他示例可知，在软件开发过程中较早修复安全缺陷将比发布后再修复安全缺陷大大节约成本。对于供应商而言，对于具有安全漏洞的软件，在其发布以后开发补丁和修补漏洞的代价被放大，而且确保应用更安全也是高成本的方法。此外，补丁并不总是由脆弱性软件的拥有者 / 用户来提供，补丁本身也可能包含更多的漏洞。^[14]我们已经看到，补丁修复了一个安全问题，但引发（或重复引发）了其他安全问题。企业并不总是能够给每一个补丁（修复）应有的重视，或者它可以不经过常规的软件开发生命周期，这会导致比打补丁或者进行全问题修复之前更多安全问题的出现。

近些年来，信息安全行业都在假定软件是安全的并且需要的一切安全控制也必备的情况下，集中研发网络和应用安全方面的技术与产品，而不是切实针对软件安全故障保护组织所

反对的软件故障开展工作。当我们回顾本书中的安全开发生命周期及相关的最佳实践时，应该明确的是，在一个全面和层次化的纵深防御安全方案中，网络和应用程序应该来得晚一点，软件自身的安全应该考虑作为信息安全生命周期的第一步，而不是最后一步。简而言之，网络和应用程序安全方案更多的是关于补偿控制，但它只能通过在源头解决安全，而且，在源头我们才能够真正地解决问题。

1.2 软件安全和软件开发生命周期

多年来，安全业界对于软件安全 and 应用安全存在着一个严重的误解。Gary McGraw 提供了关于两者之间差异的一个清晰描述：

一方面，**软件安全**是关于构建安全软件的：将软件设计成安全的；确保软件是安全的；培训软件开发人员、架构师和用户如何构建安全到软件中。另一方面，**应用程序安全**是关于保护软件和该软件发布后运行的系统，当且仅当开发完成后。^[15]

从第一个大规模针对软件的攻击开始，到 20 世纪 80 年代后期，软件安全已经走过了漫长的道路。当时软件并没有过多地考虑安全问题（如 UNIX 代码、TCP/IP 协议栈）。随着微软 Windows 以及网页（Web）的出现，攻击开始变得复杂和频繁，因此软件的安全性才逐渐得到重视。工业界开始通过各种辅助手段短期修复安全问题。这些因素推动了杀毒软件、防火墙、反间谍软件的出现。然而，真正的问题——代码是如何开发和编写的——并没有得到重视。这个问题直到最近十年来出现了 SDL 实践才得到重视。许多企业（如微软）由于软件安全缺陷的影响开始意识到通过改善软件开发实践，以确保安全的软件代码的重要性。这导致学术界和软件巨头都推荐 SDL 实践，如微软等。现在我们有理论和指导原则来帮助我们软件开发一开始就构建安全的代码，从而降低出现可能被攻击者利用的软件漏洞的可能性。

机密性、完整性和可用性是业界认为在任何软件安全开发过程中最重要的三个主要目标。开发人员保护、增强、确保这些主要目标满足安全代码定义的“高可信度”部分。开发人员能够编写非常高效的代码，这些代码易于维护并可复用。然而，如果该代码允许未经授权的用户访问应用程序的资源，那么无论该代码是否被披露，都没有第二次机会来修复它。

SDL 不应该与标准软件开发生命周期混淆。顾名思义，SDL 方法的真正目的是开发安全的软件，不一定是高质量的软件。正如 IT 维基百科定义的，“安全开发生命周期是一个软件开发过程，用来降低软件维护成本，提高软件的可靠性（涉及与 bug 相关的软件安全）”。^[16] 2002 年 1 月，微软的许多软件开发团队使用“安全推”（security push）来改进现有的安全代码。在这个方法的指导下，可信赖计算（Trustworthy Computing, TwC）团队形成了微软安全开发生命周期（Microsoft Security Development Lifecycle）。2004 年，微软 SDL 作为强制策略成为微软软件开发过程中的一个组成部分。^[17] 后来 SDL 这一术语也被其他软件公司使用，随着时间推移，SDL 既代表微软开发过程的一个组成部分，也代表前面提到的 IT 维基百科定义的开发过程。在本书中使用的术语 SDL 表示一个安全的开发过程，由以下几个方面组成：基于比较研究微软 SDL 的软件安全最佳实践，从 2004 年开始开发的替代模型，作者的经验以及对最前沿开发组织的研究有效和无效的工作，商业上对软件开发的苛刻需求再加上不断增长的需求，以在源头通过相关的、划算的和现实的软件安全实践确保代码安全。

SDL 的目标是双重的：第一个目标是减少安全漏洞和隐私问题的数量；第二个目标是减

轻遗留漏洞的严重性。有行业标准定义在软件开发中需要做些什么，比如 ISO/IEEE 定义了面向软件工程的传统软件开发方法的主要阶段。SDL 的元素通常适应性很强，很容易纳入组织的标准开发生命周期中。

静态分析和威胁建模是用于开发安全代码的工具。静态分析工具使用类似于病毒检查程序的方式，寻找代码中一组固定的模式或者规则。由于这类工具是基于定义好的规则工作的，它永远无法找到规则以外的问题，因此高级的静态分析工具允许将新的规则加入到规则库中。静态分析工具最大的好处是有能力自动识别许多常见的编码问题。但是，开发者错误带来的 bug 往往只是问题的一部分。静态分析工具无法评估设计和架构的缺陷。它们不能识别设计不当的加密库或者选择不当的算法，也不能指出可能会导致身份验证和授权混乱的设计问题。它们也不能识别内嵌在代码中密码或幻数。但是，静态分析工具可以比动态分析窥探到程序更多的阴暗角落，因为后者需要实际地运行代码。静态分析同样具备在软件达到完成程度前，就执行有效测试的潜质。在软件开发生命周期中，越早发现并管理安全风险越好。

虽然这些 SDL 实践在理论上已经很好了，但是当应用到企业中时，结果有好有坏。导致这一现状有多种原因。用 SDL 实践管理软件开发生命周期的历史并不长，因此未使用这一方法编写的（可能存在风险的）代码在软件行业的代码库中依旧占有很大比重，想要回溯并以 SDL 实践方法将这些代码再开发一遍是很困难的。软件外包和海外代工是另一个导致 SDL 很难有效落实的原因。软件开发人员往往在时间非常紧迫的情况下进行开发，导致软件的安全性被抛到脑后。在缺乏有效管理 SDL 实践的快速开发环境下，往往最后才考虑安全问题。

尽管一些安全实践对于软件 and 应用程序安全来说是通用的，比如渗透测试、源代码扫描以及安全教育，但有一个东西是任何方法都无法替代的，这就是人。这个过程中，人的因素是最关键的，成功的软件开发过程需要经验丰富的软件安全架构师和工程师。威胁建模需要应用最小授权策略和纵深防御策略，是软件开发生命周期中最为重要且任何工具都无法替代人工来完成的工作。必须收集一个系统的真正安全需求，并考虑合规性、安全性问题、合同要求、应用程序能够处理什么样的数据以及业务风险。

培训是以人为关键元素的 SDL 活动。培训有助于降低安全成本，一个好的培训方案能够促使开发团队减少软件中出现的安全问题，更高效、更划算地完成开发任务。应当强调的是，软件的安全问题不是通过某一个方法可以解决的；而是通过以人为核心，融合了管理流程和技术手段，具有整体性和纵深性的系统性方法解决的。尽管工具可以比人工更快地对大量代码进行安全性分析，但它依旧无法取代人工的作用。在可预见的未来，软件安全依然被认为是一门艺术，可以通过技术和管理流程两方面进行考量，软件安全并不是不可企及的神话，而是可以通过有经验的软件安全架构师和工程师传授。团队中有一些具有丰富经验、可以在整个开发过程中以攻击者的角度考虑问题的员工是 SDL 成功的关键因素。有些 SDL 实践者将代码安全的最佳实践和安全设计原则分开考虑；本书中，我们将它们放在一起，由经验丰富的软件安全架构师和工程师来完成这些工作。

软件的安全性并不能通过传统的软件开发管理流程得到实现，即使是那些有高质量代码开发传统的开发小组也无法不通过努力就自然而然地实现这一目标；软件安全性，应该是衡量软件代码成熟度的一个关键因素，需要集中力量专门地对待。如本书所述，安全的代码并不一定意味着高质量的代码，高质量的代码也并不一定意味着安全的代码，软件应用程序开发应该基于包含代码高质量和代码安全性的共同最佳实践。

1.3 代码的质量与安全

开发应用程序的流程，是基于代码质量和代码安全共同的最佳原则的。这些最佳原则是软件行业最佳实践的概念和设计背后的成因。为了开发经得起时间考验的安全代码，你必须学会如何将这些原则纳入到开发过程中。请记住，安全的代码不一定是高质量的代码，高质量的代码不一定是安全的代码。^[18]

安全的代码并不意味着高质量的代码：在写出安全代码之前你必须知道，如何编写高质量的代码。一个开发人员可以编写非常安全的代码，可以授权和验证用户的每一个请求，将请求内容记录到日志中。然而，如果代码连预期的结果都无法返回，即使代码写得再安全也无法使软件正常使用。软件质量的判定标准与软件安全性的判定标准不同。衡量软件安全性的是机密性、完整性和可用性，而衡量软件质量的是易用性、可重用性和可维护性。^[19]

高质量的代码并不意味着安全代码：一个开发人员可以编写高效的代码，即易于维护和可重用。但是，如果代码允许未经授权的用户获取应用系统管理的资源，它也是不能使用的。与软件质量不同的是，软件安全不是主观的。无论敏感信息是否泄露，都没有第二次机会进行修复。最终，业内认为在任何安全的软件开发过程中至上重要的三个主要目标是质量、安全和可维护性。^[20]

软件既不能只有质量没有安全，又不能只有安全没有质量。这两个属性相辅相成，两者兼顾才能提升软件产品的完整性和市场价值。好的开发人员应该能够清楚地知道影响软件质量的因素有哪些，以及如何在代码编写中实现它们。同样，优秀的开发人员应该知道他们开发的软件可能面对哪些攻击，以及软件最薄弱的地方；如果代码允许未经授权的用户访问应用程序管理的资源，那么无论该代码是否被披露，都没有第二次机会修复它。^[21]

1.4 SDL 三个最重要的安全目标

任何一个合格的开发人员都可以编写非常高效的、可维护的和易于重用的代码；但是，如果该代码允许未经授权的用户访问应用程序管理的资源，该代码就是不安全的。遗憾的是，在软件开发生命周期中，安全性仍然是经常被忽视或者投入最少的方面。信息安全界认为SDL最低也是最为重要的三个目标是：

1. 机密性
2. 完整性
3. 可用性

这三个目标统称为 C.I.A (Confidentiality, Integrity, Availability)。人们普遍认为，在软件开发生命周期中，开发者通过公认有效的方法对 C.I.A. 进行保证、增强、保护，就认为代码是高可信和安全的。

信息安全性、机密性、完整性和可用性在《44 U.S.C., Sec. 3542》中的定义如下所示。

信息安全性：保护信息和信息系统，避免未经授权的访问、使用、泄露、中断、修改或破坏，以确保信息的机密性、完整性和可用性。

机密性：对信息的访问和泄露进行限制，包括保护个人隐私和专有信息的手段。

完整性：防止不正确的信息修改或销毁，包括确保信息不可抵赖性和真实性。

可用性：保证信息可以实时使用。^[22]

机密性、可用性和完整性合在一起就是信息安全。

机密性是通过阻止非授权用户（人或软件）访问授权信息实现的。如果保证机密性，软件就会认为是可信的。授权和认证是机密性的两个属性，授权确保用户具有相应的角色和特权获取数据，认证确保用户确实是他们所声称的，并且认证数据来自合适的地方。应用程序的完整性包括数据的接受、传输和存储。数据必须保证没有被未授权的用户修改，数据从输入点到数据库，以及从数据库返回到输出点都应该是可靠的。数据加密、数字签名和公钥是一些保证完整性的方法。除了限制宕机时间外，可用性还包括系统或软件正常工作的时间占总工作时间的比例。作为软件安全的重要组成部分，缺乏机密性、可用性和完整性将导致产品声誉及销售额受损。最后，在好的业务流程中，软件的安全性、软件的质量是同等重要的。

1.5 威胁建模和攻击面验证

威胁建模和攻击面验证也许是 SDL 中最耗时、最不易理解和困难最大的部分。它需要软件安全架构师——安全团队中经验最为丰富的专家来完成。简而言之，威胁建模的目的是理解系统的潜在威胁，确定风险，建立适当的应对措施（有什么风险？风险严重性有多大？如何消除风险？）。当在项目生命周期的早期正确地进行威胁建模后，就可以在代码提交前发现软件设计的安全问题。威胁建模使问题在软件开发生命周期的早期就得到了解决，从而节省了大量成本。威胁建模还有助于企业管理软件风险，并提供将技术风险转换为业务影响的能力。基本原则是，在软件生命周期中越早识别和管理安全风险越好。

威胁建模需要从黑客的角度深入思考，以这样一个软件行业中特殊的专业知识来思考软件中所有可能被攻击者攻击的薄弱环节。因此，这是一个与其他方法稍微不同的测试方法。尽管专业的质量控制人员可以做一些安全测试并发现一些典型漏洞，但是他们一般是从客户的角度来思考问题的，而不是从黑客的角度。通常企业内部没有这样的专人人员，需要聘请第三方机构来做这项工作。

美国一家软件数据分析中心（Data and Analysis Center for Software, DACS）2008 年 10 月的报告“用增强软件开发生命周期的方法开发安全的软件产品：软件保障参考指南”（Enhancing the Development Lifecycle to Produce Secure Software: A Reference Guidebook on Software Assurance）定义了一个软件密集型系统的威胁，即“只要一个系统外部允许输入，任何参与者、代理、环境因素或者事件都有可能给系统、数据或者资源造成危害的情况”。^[23] 威胁可以根据其意向进行分类。例如，威胁可能是无意的、故意的但非恶意的或者恶意的；恶意的威胁被界定为故意的。虽然所有三个类别的威胁都有可能破坏软件安全性，但只有恶意的威胁被认为是攻击。DSCS 报告还指出：

针对软件的大多数攻击充分利用了软件的一些漏洞或弱点；由于这个原因，“攻击”（attack）经常与“利用”（exploit）互换使用，尽管 BuildSecurityIn 攻击模式词汇表已明确区分了这两个名词，攻击指的是针对目标软件本身的，而利用攻击针对的是有缺陷的机制（例如，一种技术或恶意代码）。

软件建模是将设想的软件与环境的交互以直观的方式表现出来的方法。模型对预期环境描述得越精确，模型起到的作用就越大。因此，安全软件设计和开发得益于明确安全威胁后的建模。正如 DACS 2007 年软件安全保证先进报告（Software Security Assurance State-of-the-

Art Report, SOAR) 所描述, “建模的主要问题是要做好; 要全面; 针对建模的结果该如何应对 (例如, 如何将分析的结果转化为可量化或其他方便使用的描述点)”。结合威胁和建模的概念, 该报告定义威胁建模为 “……一种用来评估和记录与应用程序相关的安全风险弱点的系统方法。要求开发团队站在攻击者的角度攻击应用程序, 从而确定高风险的组件”。^[24]

在时间和资源都有限的情况下, 测试一个应用程序中所有的代码是不可能的。但是, 测试至少要涵盖应用程序的入口点和出口点, 这些暴露在外的部分很可能被攻击者利用, 入口点和出口点统称为应用程序的攻击面。可访问性会增加软件的攻击面。例如, 限制管理员只能在本地登录的软件的攻击面, 小于允许匿名用户远程访问的软件的攻击面。

通过覆盖应用程序中的所有代码路径的测试可达到对攻击面全面测试的目的。攻击面的元素可以通过扫描工具来识别, 如使用端口扫描工具发现系统开启的端口, 使用代码分析工具定位代码中接收输入和发送输出的部分。对于定制的应用程序, 甚至需要自己开发工具来确认程序的入口点。最小攻击面通常在软件开发生命周期的早期进行定义, 并在之后的阶段进行测试。在测试之前正式定义和测试攻击面往往是有帮助的。正如本书后面讨论的, 尽管工具在此阶段会有帮助, 但人工仍然是必需的, 比如经验丰富的软件安全架构师。

1.6 本章小结: 期望从本书中学到什么

软件安全只与软件开发团队使用的最佳实践的质量和相关性紧密相关。软件安全必须从一开始就构建。软件安全是设计开始阶段的重要组成部分, 并包含在每一个后续的发展阶段, 贯穿于完整的系统中。通过 SDLC 的安全增强流程尽早地解决安全漏洞, 要比系统产品化后再修复和纠正这些安全漏洞成本低得多。这将会大大减少软件发布后需要修补的漏洞的数量, 这些漏洞会损坏公司的声誉从而降低公司的经济效益。今天, 我们看到对软件开发的安全需求日益增加, 安全性需求、设计和防御原则已融入到传统 SDLC 中, 最重要的是, 选择在 SDLC 的所有活动中都能满足这方面需求的安全开发实践。

我们曾在 “财富 500 强” 企业工作, 并经常看到 SDL 实践中失败的例子。在这本书中, 我们采取以经验为基础的方法, 运用最佳的 SDL 模型组件, 并在一个 SDL 软件安全最佳实践模型和框架下处理上述问题。我们将首先概述安全开发生命周期, 然后将 SDL 最佳实践映射到软件开发生命周期的模型, 描述如何使用它来建立一个成熟的 SDL 项目。尽管安全性并没有在目前的软件行业得到重视, 但是我们相信, 在软件开发流程中融入安全性的增强是必要的、可以实现的, 也是重要的, 我们相信本书介绍的软件安全最佳实践和模型将为所有读者明确这一点, 无论你是经营者、管理者还是实践者。

参考文献

1. President's Information Technology Advisory Committee (2005), *Cybersecurity: A Crisis of Prioritization*, Executive Office of the President, National Coordination Office for Information Technology Research and Development, 2005, p. 39. Retrieved from http://www.nitrd.gov/Pitac/reports/20050301_cybersecurity/cybersecurity.pdf.
2. Ibid.
3. Aras, O., Ciaramitaro, B., and Livermore, J. (2008), “Secure Software Development—The Role of IT Audit,” *ISACA Journal*, vol. 4, 2008. Retrieved

- from <http://www.isaca.org/Journal/Past-Issues/2008/Volume-4/Pages/Secure-Software-Development-The-Role-of-IT-Audit1.aspx>.
4. U.S. Department of Homeland Security (2006), *Security in the Software Lifecycle: Making Software Development Processes—and Software Produced by Them—More Secure*, DRAFT Version 1.2, p. 13. Retrieved from <http://www.cert.org/books/secureswe/SecuritySL.pdf>.
 5. Schwartz, M. (2012), “10 Security Trends to Watch in 2012.” Retrieved from <http://www.informationweek.com/security/vulnerabilities/10-security-trends-to-watch-in-2012/232400392>.
 6. Parizo, E. (2012), “To Get Help with Secure Software Development Issues, Find Your Own Flaws.” Retrieved from <http://searchsecurity.techtarget.com/news/2240129160/To-get-help-with-secure-software-development-issues-find-your-own-flaw>.
 7. Microsoft Corporation (2012), Security Development Conference 2012 webpage, May 15–16, 2012, Washington, DC. Retrieved from <https://www.securitydevelopmentconference.com/main.aspx>.
 8. blackhat.com (2013), Black Hat USA 2012 Conference webpage, July 21–26, 2012, Las Vegas, NV. Retrieved from <http://www.blackhat.com/html/bh-us-12>.
 9. rsaconference.com (2013), RSA 2013 Conference USA webpage, February 25–March 1, 2013, San Francisco, CA. Retrieved from <http://www.rsaconference.com/events/2013/usa>.
 10. securitydevelopmentconference.com (2013), Security Development Conference 2013, May 14–15, 2013, San Francisco, CA. Retrieved from <http://www.securitydevelopmentconference.com>.
 11. Boehm, B., and Papaccio, P. (1998), “Understanding and Controlling Software Costs,” *IEEE Transactions on Software Engineering*, vol. 14, no. 10, October 1988, pp. 1462–1477.
 12. Boehm, B., and Basili, V. (2001), “Software Defect Reduction Top 10 List,” *Computer*, vol. 34, no. 1, January 2001, pp. 135–137.
 13. Meftah, B. (2008), “Business Software Assurance: Identifying and Reducing Software Risk in the Enterprise,” 9th Semi-Annual Software Assurance Forum, Gaithersburg, MD, October 2008. <https://buildsecurityin.us-cert.gov/swa/downloads/Meftah.pdf>.
 14. Viega, J., and McGraw, G. (2006), *Building Secure Software: How to Avoid Security Problems the Right Way*, Boston: Addison-Wesley.
 15. McGraw, G. (2006), *Software Security: Building Security In*, Boston: Addison-Wesley, p. 20.
 16. IT Law Wiki (2012), “Security Development Lifecycle Definition.” Retrieved from http://itlaw.wikia.com/wiki/Security_Development_Lifecycle.
 17. Microsoft Corporation (2012), “Evolution of the Microsoft SDL.” Retrieved from <http://www.microsoft.com/security/sdl/resources/evolution.aspx>.
 18. Grembi, J. (2008), *Secure Software Development: A Security Programmer's Guide*, Boston: Course Technology.
 19. Ibid.
 20. Ibid.
 21. Ibid.
 22. United States Government (2006), 44 U.S.C., SEC. 3542: *United States Code, 2006 Edition*, Supplement 5, Title 44; CHAPTER 35 – COORDINATION OF FEDERAL INFORMATION POLICY, SUBCHAPTER III – INFORMATION SECURITY, Sec. 3542 – Definitions. Retrieved from <http://www.gpo.gov/fdsys/pkg/USCODE-2011-title44/pdf/USCODE-2011-title44-chap35-subchapIII->

- sec3542.pdf.
23. Goertzel, K., et al., for Department of Homeland Security and Department of Defense Data and Analysis Center for Software (2008), *Enhancing the Development Life Cycle to Produce Secure Software: A Reference Guidebook on Software Assurance*, Version 2, October 2008. Retrieved from https://www.thedacs.com/techs/enhanced_life_cycles.
 24. Goertzel, K., et al. (2008), *Software Security Assurance: State-of-the-Art Report (SOAR)*, July 31, 2008. Retrieved from <http://iac.dtic.mil/iatac/download/security.pdf>.

安全开发生命周期

本章首先介绍使用安全开发生命周期（Secure Development Lifecycle, SDL）来克服软件安全中挑战的相关概念。接下来会进一步讨论与管理 and 克服软件安全挑战相关的模型、方法论、工具、人才及度量标准。最后将讨论把 SDL 及其相关的最佳实践应用到一个通用的软件开发生命周期（Software Development LifeCycle, SDLC）中，这将是接下来 6 章的主题，随后一章讲述如何将 SDL 最佳实践应用到几个最流行的软件开发方法中。

现在仍然有必要将更好的静态和动态测试工具，以及形式化安全方法融入 SDLC 中，这在大多数软件开发机构力所能及的范围之内。在过去的十年左右，除了大多数资源丰富的公司以外，主要的 SDL 模型几乎对所有公司来说是遥不可及的。本书的目标是：基于最少的资源和最佳实践创建 SDL，而不是所需资源超出大多数软件安全团队所能接受的范围。

2.1 克服软件安全中的挑战

如第 1 章所述，SDL 是软件安全演化的关键步骤，有助于人们重视构建安全的软件开发生命周期。过去，软件产品的利益相关者并不把软件安全作为一项高度优先的事项。人们认为，一个安全的网络基础设施将会提供针对恶意攻击所需的保护级别。然而，最近几年，已经证明单纯的网络安全不足以防止这种攻击。用户已经可以通过相关技术成功渗透有效的渠道认证，相关技术包括跨站点脚本（Cross-Site Scripting, XSS）、结构化查询语言（Structured Query Language, SQL）注入和缓冲区溢出漏洞利用技术等。在这种情况下，系统资产被泄露，数据和组织的完整性也遭到破坏。安全行业一直试图尝试通过应急措施来解决软件安全问题。首先是平台安全（OS 安全），然后是网络 / 周边安全，以及现在的应用程序安全。我们确实需要深度防御来保护我们的资产，但从根本上它是一个软件安全漏洞，需要通过 SDL 方法进行修复。

SDL 拥有开发行业、政府标准和最佳实践强化软件所需的所有活动与安全控制，它们也是 SDL 的基本组件。为了能够真正阻止、识别和减轻已开发系统中的可利用漏洞，经验丰富的工作人员、安全软件的策略和管制要求是必需的。

如果不满足 SDL 中的最小活动要求，将会给内部和外部威胁提供误用系统资产的机会。安全不再单纯是一个网络要求，它现在是一个信息技术（IT）要求，其中包括所有的软件开发意图分发、存储和处理信息。为了保护客户的隐私和他们的重要信息，组织必须实施最高标准的开发以保证发布最高质量的产品。

SDL 程序的实现，能够保证良好的企业软件设计和开发固有的安全，而不是在产品发布后。使用 SDL 方法能够产生切实的好处，例如保证所有的软件版本符合最低安全标准，所有的利益相关者支持和强制执行安全准则。在开发周期的早期漏洞更容易被修复且成本更低，此时给信息安全团队提供一个系统化的方法在开发过程中协同消除软件风险。

最著名的 SDL 模型是可信计算安全开发生命周期（或 SDL），微软已经将它应用到需要承

受恶意攻击的软件开发中。微软的 SDL^[1] 已经发展了十多年，被认为是排名前三的成熟模型。其他受欢迎的 SDL 模型有 Cigital 公司的软件安全触点模型（Software Security Touchpoints model）^[2]、OWASP SDL^[3] 和思科的安全开发生命周期（Cisco Secure Development Lifecycle, CSDL）。^[4]

微软的 SDL 也有一个安全开发生命周期（SDL）的优化模型^[5]，由微软以外的开发机构设计，旨在促进渐进的、一致的和具有成本效益的 SDL 实现。该模型帮助那些负责将安全和隐私集成到他们组织的软件开发生命周期的人，评估其当前的状态，并逐步推进组织使用可靠的微软程序以生产更安全的软件。

SDL 优化模型使得开发经理和 IT 决策者能够评估开发中安全的状态。基于此，他们可以为降低客户风险创建一个视图和路线图，以一种具有成本效益、一致和渐进的方式编写更加安全与可靠的软件。通过推进 SDL 优化模型的成熟度等级，组织管理层对 SDL 目标和结果的承诺将由初步接受提高为强有力的授权。^[6]

2.2 软件安全成熟度模型

近年来，两个非常流行的软件安全成熟度模型已先后开发并继续快速地成熟。一个是 Cigital 的内置安全成熟度模型（Building Security in Maturity Model, BSIMM）^[7]，另一个是 OWASP（Open Web Application Security Project, 开放 Web 应用安全项目）的开放软件保证成熟度模型（Software Assurance Maturity Model, SAMM）^[8]。BSIMM 是一项面向真实软件安全措施的研究，帮助你确定软件安全措施和如何组织工作时间。BSIMM 是 Cigital 开发的一系列最佳实践，分析 9 个领先的软件安全措施的真实数据，并基于成功的公共区域创建框架。有 12 个实践组织为 4 个域。这些实践都用于组织 109 个 BSIMM 活动（BSIMM 4 共有 111 个活动）。

通过研究这 9 个措施都在干些什么，BSIMM 的创造者能够建立一个最佳实践模型，该模型分解为软件制造商可以遵循的 12 个类：

1. 策略和度量标准
2. 符合性和政策
3. 培训
4. 攻击模型
5. 安全特征和设计
6. 标准和要求
7. 架构分析
8. 代码审查
9. 安全测试
10. 渗透测试
11. 软件环境
12. 配置和漏洞管理^[9]

BSSIM 的第 4 个版本发布于 2012 年 9 月 18 日，它的一些要点如下。

- BSIMM 项目中，除了原有的 109 个活动之外，首次观察到新的活动，因此未来模型增加了两个新的活动：模拟软件危机和自动化恶意代码检测。

- BSIMM4 包括来自 12 个垂直行业的 51 家公司。
- BSIMM4 相对于 BSIMM3 增长了 20%，比 2009 年版大十几倍。
- BSIMM4 数据集有 95 个不同的测量（有些公司多次测量，一些公司由多个部门单独测量并得到一个公司评分）。
- BSIMM4 继续表明，领先的公司雇用的每 100 个开发人员中平均有两名全职的软件安全专家。
- BSIMM4 介绍了与 2039 名保证某软件安全的开发人员共事的 974 名软件安全专业人士的工作，该软件由 218 286 名开发人员完成。^[10]

OWASP 的软件保证成熟度模型（SAMM）是一个灵活的规范性框架，用于把安全性融入软件开发组织。通过覆盖多于典型的 SDLC 基础模型的安全性，SAMM 使得组织能够自行评估其安全保障程序，然后使用建议的路线图在组织面对的特定风险方面有所改进。除此之外，SAMM 能够创建记分卡，记录在典型的管理、开发和部署业务功能过程中组织在安全开发方面的有效性。记分卡也能够在组织内部实现管理，通过迭代建立安全保证程序来说明量化的改进。^[11]

2.3 ISO/IEC 27034：信息技术、安全技术、应用安全

2011 年，国际标准化组织（International Standards Organization, ISO）/ 国际电工委员会（International Electrotechnical Commission, IEC）发布 ISO / IEC27034-1:2011 中 6 个应用安全标准的第 1 部分。^[12] 该标准提供了一个简洁并且国际公认的方式来获得厂商 / 供应商软件安全管理流程的透明性。为了配合不同的工程组织，该标准设计具有足够的灵活性，但为了解决现实世界的风险，它又足够具体。虽然它还尚未完成，但它即将完成并很有可能会类似于 ISO / IEC27001 的 IT 安全标准，即客户、合作伙伴和他们的工程团队将会期望的规范。作为一个来自于国际机构而不是厂商的软件安全标准，它还没有绑定到特定的技术。在撰写本书时，第 2 ~ 6 部分仍处于工作草案阶段，由以下 5 部分组成：第 2 部分，组织规范性框架；第 3 部分，应用安全管理流程；第 4 部分，应用安全验证；第 5 部分，协议和应用安全控制数据结构；第 6 部分，具体应用的安全指导。^[13] 多年来，当组织（和他们的客户）开始关注信息安全时，安全和规范行业提出了大量的证明、认证和方法。这些标准 / 证明都声称自己是独特的，因为它们能够衡量一个组织的安全状态。竞争和市场宣传导致混乱，不同的组织规范着不同的认证。作者已经看到组织正在推动他们的客户（在大多数情况下，其他公司）采用他们推荐的认证。对于“财富 500 强”企业来说，这意味着获得多重的证明 / 认证作为安全状况的一个证据。由于大多数证明 / 认证集中关注“合规监控”或“基于策略的安全”，这并没有帮助。随着诸如 SOX、GLBA、Safe Harbor 和 HIPAA 等规则更添混乱，情况将变得更糟。公司经常要进行一系列认证，分别用于规范、安全、隐私、信用卡、人身安全等。

ISO/IEC 制定了 ISO/IEC 27001 标准（纳入 ISO/IEC 17799，这一直是信息安全事实上的 ISO 标准）。它是一个信息安全管理体系（Information Security Management System, ISMS）标准，通过制定一个管理体系获得正式管理控制下的信息安全。它规定了一个组织采用该标准时需要满足的具体要求。该标准从整体上解决了信息安全问题，囊括了从物理安全到规范的所有内容。行业踊跃采用该实践，ISO/IEC 27001 是当今面向信息安全管理体系（ISMS）的领先标准。大多数其他标准的控制策略都可以映射回 ISO/IEC 27001 标准中。这使得组织能够

在一个标准下加强各种安全力度，寻求整体安全考虑下的单一框架，并以一种一致的方式收集度量标准来衡量和管理一个组织的安全。

在 ISO/IEC 27001 标准出现的几年前，作者看到与信息安全类似的软件安全（和 SDL）已经出现。有多种 SDL 方法（开放的和专有的），每一个都自称比下一个好。混乱阻碍了组织中实现软件安全的最佳方式。对一个组织应用任何一个框架，要求组织采用不同的过程或者定制一个使用他们环境的 SDL 框架。随着 ISO/IEC 27034 标准的出现，作者看到软件安全标准 / 框架开始整合，就像 ISO/IEC 27001 标准对于信息安全的影响。即使处于起步阶段，也应该意识到 ISO/IEC 27034 标准的重要性。微软已经宣布其 SDL 方法将会遵守 ISO/IEC 27034-1 标准。^[14] 我们期望在不久的将来能够看到其他框架类似的结果。

ISO/IEC 27034 标准提供帮助组织内嵌过程（包括应用程序生命周期）安全方面的指导，有助于环境中运行安全的应用程序。它是一种基于风险的框架，通过流程整合 / 改进来管理应用程序，持续不断地提高其安全性。这需要过程方法的设计。

作者推荐的 SDL 框架可以映射到 ISO/IEC 27034 框架中。我们将会在附录 A 中列出相关的映射关系。

2.4 其他 SDL 最佳实践的资源

还有其他 SDL 最佳实践的资源，下面介绍其中几种最流行的资源：

2.4.1 SAFECODE

卓越代码软件保障论坛（Software Assurance Forum for Excellence in Code, SAFECODE）是一个非营利性组织，致力于通过先进有效的软件保证方法，增加信息、通信技术产品和服务的可信性。SAFECODE 是一项全球性的、行业主导的工作，确定和推广开发提供更安全可靠的软件、硬件和服务的最佳实践。它意味着提供一系列基础的安全开发实践，在不同的开发环境下由 SAFECODE 成员实际实现的软件安全已得到有效改善。通过对成员个人软件安全工作的不断分析，我们确定这些是 SAFECODE 成员采用的“练习实践”。把这些方法集合起来并分享给更大的社区，SAFECODE 希望在这个行业理论定义的最佳实践之外，描述一系列软件工程实践，其已被证明能够改善软件安全，并当前使用在领先的软件公司中。^[15,16]

2.4.2 美国国土安全软件保障计划

自 2004 年以来，美国国土安全部（Department of Homeland Security, DHS）软件保障计划已资助了网站内置安全（Build Security In, BSI）的开发。^[17] BSI 的内容基于以下原则：软件安全本质上是一个软件工程问题，必须在整个 SDLC 中以一种系统的方式进行管理。

美国国土安全部国家网络安全办公室（National Cyber Security Division, NCSD）的软件保障（Software Assurance, SwA）计划旨在减少软件漏洞，最大限度地减少开发，以及改善可信软件产品的日常开发和部署。与开放政府指令一致，该计划促使公共和私营部门合作，包括开发、发布和推广实用指南与工具的使用；促进更安全、更可靠软件产品的投资。美国国土安全部软件保障计划与私营部门、学术界以及其他联邦部门和机构合作，通过活动加强软件生命周期过程和技术的安全，比如软件保障论坛，是由美国国防部（Department of Defense, DoD）和美国国家标准与技术研究所（National Institute of Standards and Technology, NIST）

共同赞助的。由美国国土安全部国家网络安全办公室和美国国家安全局（National Security Agency, NSA）资助的一项关键倡议是共同缺陷枚举（Common Weakness Enumeration, CWE）。CWE 是 DHS 和 NSA，以及软件社区（包括政府、私营部门和学术界）的共同成果，MITRE 公司提供技术指导和项目协调。超过 800 个软件缺陷已经被确定和编目。超过 47 个产品和服务已经以一种兼容的方式使用 CWE。为了减少多次被利用的编程错误，SANS 协会（软件保障论坛的积极参与者），整理了前 25 个 CWE。SANS 提出了关注前 25 个 CWE 的想法，这种努力代表了团体合作，优先考虑最可能被利用（使软件容易遭受攻击或出现故障）的结构。这促进了 DHS 共同赞助的 CWE 并巩固了 SANS 于 2001 年创立的“Top XXX”品牌，开始是 Top 10——组织应该优先解决的第一份安全问题列表。^[18]

CWE 是 NCSD 软件保障计划的重要组成部分。这份错误清单使得 CWE 成为实用的、可操作和可度量的焦点，将会帮助人们制作并展示真正的进展。公共与私营部门的合作构成了 NCSD SwA 的计划的基础。CWE 是一直倡导的公私合作的一个范例。与开放政府指令一致，SwA 计划的 CWE 赞助使得团队参与、协作和透明成为可能。CWE 提供了可利用软件架构必要的表征；从而在软件交付并投入运营之前，能够更好地教育和培训程序员如何消除常见的错误。这与 BSI 方法相互配合，使得软件更安全地开发，从而避免长远来看的安全问题。CWE 提供了理解剩余风险的标准，从而使得关心软件安全的供应商和消费者能够做出更明智的决策。^[19]

2.4.3 美国国家标准与技术研究院

美国国家标准与技术研究院（National Institute of Standards and Technology, NIST）主要向政府和企业信息安全社区提供研究、信息和工具。以下是 NIST 贡献给软件安全社区的一些关键领域。

NIST 的软件保障度量和工具测量（Software Assurance Metrics And Tool Evaluation, SAMATE）项目致力于开发软件工具测量方法（测量工具和技術的有效性，确定工具和方法之间的差异）以提高软件保障。该项目支持美国国土安全部的软件保障工具和研发需求识别计划——特别是第 3 部分：技术（工具和要求）、识别、增强和软件保障工具的开发。SAMATE 项目的范围很广，从操作系统到防火墙，从 SCADA 到 Web 应用程序，从源代码安全分析到正确的实施方法都包含。

NIST 特别出版物（Special Publication, SP）800-64,《系统开发生命周期中的安全考虑》，已在完善中，以协助联邦政府机构把重要的信息技术安全步骤整合到他们现有的 IT 系统开发生命周期中。这个安全考虑指导原则适用于除了国家安全系统之外所有的联邦 IT 系统。该文件旨在作为一个参考资源，而不是一个教程，在整个系统的开发过程中应该与其他所需的 NIST 出版物一起使用。

美国国家漏洞数据库（National Vulnerability Database, NVD）是基于标准的漏洞管理数据使用安全内容自动化协议（Security Content Automation Protocol, SCAP）表示的美国政府资料库。这些数据实现了漏洞管理、安全测量和合规的自动化。NVD 包括安全检查表数据库、安全相关的软件缺陷、错误配置、产品名称和影响度量标准。^[22] NVD 公共漏洞评分系统（Common Vulnerability Scoring System, CVSS）提供了一个开放的框架，用于表示 IT 漏洞的特征和影响。其量化模型确保可重复的精确测量，同时用户能够看到用来生成分数的基本漏洞

特征。因此, CVSS 非常适合作为标准测量系统用于产业、企业和政府中, 它们需要准确和一致的漏洞影响分数。CVSS 两个常见的应用是: 优先漏洞修复活动; 计算某一系统中发现漏洞的严重程度。NVD 提供几乎所有已知的漏洞的 CVSS 分数。特别是, NVD 支持面向所有 CVE 漏洞的 CVSS Version 2 标准。NVD 提供 CVSS “基础分数”, 代表每一个漏洞的先天特征。它目前不能准确提供“瞬时分数”(由于漏洞外部事件的变化, 分数随时间而改变)。然而, NVD 确实提供了一个 CVSS 分数计算器, 可以计算瞬时分数, 甚至计算环境得分(反映该漏洞对组织的影响)。该计算器支持美国政府机构依据 FIPS 199 系统排名自定义漏洞影响分数。我们将在本书后面的章节讨论管理软件安全 CVSS 分数的使用。^[23]

2.4.4 MITRE 公司公共计算机漏洞和暴露

MITRE 公司公共计算机漏洞和暴露 (Computer Vulnerability and Exposure, CVE) 是一个被广泛认同的信息安全漏洞和暴露的清单。CVE 的目标是使用这个“共同枚举”能够更容易地在独立的漏洞功能(工具库、存储库和服务)间共享。信息安全漏洞是可以直接被黑客利用进而访问系统或网络的软件错误, 请参阅 CVE 网站 Terminology 页面详细了解在 CVE 中如使用该术语。信息安全暴露是允许访问信息的软件错误, 或者被黑客利用作为跳板访问系统或网络的能力。采用通用的标识符可以更容易地在不同的数据库、工具和服务中共享数据, 直到 1999 年 CVE 才出现, 它不容易集成。如果安全能力报告包含 CVE 标识符, 就可以快速、准确地访问一个或多个兼容 CVE 的独立工具、服务和存储库中的信息, 以便处理这个问题。借助 CVE, 工具和服务能够“对话”(即交换数据)。因为 CVE 提供了用于评估工具覆盖面的基线, 所以你会确切知道彼此覆盖了什么。这意味着可以决定哪一个工具是最有效和最适合组织需求的。总之, CVE 兼容的工具、服务和数据库将会带来更好的覆盖面、更容易的互操作性和增强的安全性。

Bugtraq ID 是面向一个商业运作的漏洞数据库(使用在安全公告和警报中)和 Bugtraq 邮件列表讨论的标识符。CVE 标识符来自国际信息安全的努力, 是公开可用的并且是免费使用的。CVE 标识符的唯一目的是提供一个通用名称。为此, CVE 标识符经常被研究人员以及安全工具、网站、数据库和服务制造商使用, 作为识别漏洞和与其他也使用 CVE 标识符的存储库交联的标准方法。对于任何给定的漏洞或暴露, CVE 标识符会提供标准化的标识符。知道这种标识符, 就可以快速、准确地访问 CVE 兼容的多个信息源中相关问题的信息。例如, 如果你拥有一个其报告包含 CVE 标识符引用的安全工具, 你就可以获得另一个兼容 CVE 的数据库中的修复信息。CVE 还提供了一个评估工具覆盖范围的基线。

CVE 列表反馈给美国 NVD, 该数据库以包含在 CVE 条目中的信息为基础, 给每一个 CVE 标识符提供增强型信息(比如: 修复信息、严重程度打分、影响指数)。NVD 也提供了高级搜索功能, 如依据个人 CVE-ID、操作系统、供应商名称、产品名称和 / 或版本号、漏洞类型、严重程度、相关开发范围和影响。

CVE 由美国国土安全部 NCSD 资助。US-CERT 是 NCSD 的业务部门。US-CERT 将 CVE 标识符合并到其安全顾问中(只要有可能), 并提倡美国政府和所有信息安全社区的成员使用 CVE 和 CVE 兼容的产品与服务。MITRE 公司负责维护 CVE 和公共网站, 管理兼容性计划, 并向编辑部提供公正的技术指导, 以确保 CVE 服务于公众利益。^[24, 25]

2.4.5 SANS 研究所高级网络安全风险

SANS 研究所高级网络安全风险，之前的 SANS 20 个最关键的互联网安全漏洞，是互联网安全中最关键问题领域的一个共识列表，如果系统上存在这些漏洞就需要立即修复。纠正这些安全缺陷的步骤说明和更多信息的链接也作为该列表的一部分。SANS 列表包括 CVE 标识符来唯一标识它描述的漏洞。这有助于系统管理员使用 CVE 兼容的产品和服务，使他们的网络更安全。^[26, 27, 28]

2.4.6 美国国防部网络安全与信息系统信息分析中心

2012 年 9 月，软件数据与分析中心（Data & Analysis Center for Software, DACS）、信息保障技术分析中心（Information Assurance Technology Analysis Center, IATAC）和建模与仿真信息分析中心（Modeling and Simulation Information Analysis Center, MSIAC）合并，成立了网络安全与信息系统信息分析中心（Cyber Security and Information Systems Information Analysis Center, CSIAC）。CSIAC 是美国国防技术信息中心（Defense Technical Information Center, DTIC）资助的 8 个信息分析中心之一，执行必要的基本操作中心（Basic Center of Operations, BCO）功能，实现网络安全、信息保障、知识管理和信息共享、软件密集型系统工程以及建模和仿真的使命和目标，其适用于美国国防部研究开发测试和评估（Research Development Test and Evaluation, EDT&E）以及收购社区的需求。^[29] 过去，DACs 已经为社区提供了一些关于软件安全和 SDL 的优秀文档，尤其是，“提高开发生命周期以生产安全软件：软件保障参考指南”（2008）^[30] 和联合 IATAC/DACS 的报告“软件安全保障：技术现状报告”（SOAR）（2008）^[31]，我们期望他们能够在 CSIAC 的保护伞下继续这样做。

2.4.7 CERT、Bugtraq 和 SecurityFocus

除了上面讨论的资源以外，卡内基梅隆计算机应急响应小组（Computer Emergency Readiness Team, CERT）^[32]、Bugtraq^[33] 和 SecurityFocus^[34] 是读者应该知道的其他 3 个资源。

CERT 提供安全漏洞的及时警报和漏洞的每周总结公告（CERT 网络安全公告）。公告中的信息包括 CVSS 分数和 CVE ID，后者用来唯一标识漏洞。汇编结果建立在过去的一周中 NIST NVD^[35] 所记录的基础之上。本书后面的章节将详细讨论 CVSS 打分过程。

Bugtraq 是一种电子安全邮件列表，提供安全漏洞信息和来自供应商的安全公告及通知。该列表通常包含额外的信息，例如漏洞利用示例和已发现问题的修复。Bugtraq 是 SecurityFocus 安全门户网站的一部分，其目前由赛门铁克（Symantec）所有。Bugtraq 是许多通过 SecurityFocus 的安全邮件列表之一。还有其他一些有用的邮件列表，如专用于微软、Linux、IDS 和意外事件的邮件列表。

2.5 关键工具和人才

如同所有的安全任务一样，无论它们的方法是攻击还是防御，总要有成功所需的过程、技术和人力的一个融合。迄今为止，可用于软件安全的过程和模型已在本节中讨论了。有两个要素：一个是技术（工具）方面，它在软件安全方面提供帮助或制造麻烦；另一是人力（人才）方面。

2.5.1 工具

三个主要的工具是 SDL 的基础：模糊测试、静态和动态分析工具。本书后面的章节将详细介绍它们在 SDL 使用方面的最佳实践，一个高层次的概述如下。

2.5.1.1 模糊测试

模糊测试（Fuzz testing/Fuzzing）是一种自动或半自动化的黑盒软件测试技术，它为某一计算机软件程序输入提供了无效、意外或者随机数据。换句话说，它通过使用一种自动化的畸形 / 半畸形数据注入方式发现 bug 或者安全缺陷。软件程序的输入用于监控异常返回，如崩溃、出错的内置代码断言和潜在的内存泄漏。模糊测试已成为测试软件或计算机系统安全问题的一个关键要素。相对于其他工具，模糊测试一个明显的优势是，其测试设计非常简单并且与系统行为无关。

模糊测试是软件安全的关键要素，必须嵌入到 SDL 中。有很多供应商在这个领域可供选择，有的开发商甚至开发他们自己的工具。两个流行的模糊测试工具是 Codenomicon^[36]和 Peach Fuzzing Tool^[37]，Codenomicon 是市场上可用的最成熟的模糊测试工具之一，Peach Fuzzing Tool 是更受欢迎的开源工具之一。正如在本书后面章节所看到的，模糊测试工具在 SDL 中使用的时机是至关重要的。还应当注意的是，模糊测试适用于安全和质量保障测试。

在很多软件开发计划中，已认为模糊测试既是一个关键要素，也是一个主要的不足之处，因此它现在是美国国防部的信息保障认证和认定程序（Defense Information Assurance Certification and Accreditation Process, DIACAP）的要求。

2.5.1.2 静态分析

静态程序分析是指在不实际运行程序的条件下，进行计算机软件分析的方法。它主要针对特定版本的源代码，也有些静态程序分析的对象是目标代码。与此相反，动态分析需要在程序运行时才能进行。静态分析是通过一个自动化的软件工具进行的，不应与人工分析或者软件安全架构评审相混淆，其中涉及人工代码评审、程序认识和理解。如果使用得当，静态分析工具相对于人工静态分析具有明显的优势，即分析能够更加频繁地进行，安全知识普遍优于标准的软件开发人员。它还给予经验丰富的软件安全架构师或者工程师充足的时间，只需在绝对必要的时候他们出现即可。

静态分析，也称为静态应用安全测试（Static Application Security Testing, SAST），在一个项目的开发或质量保障阶段识别漏洞。它提供了行代码级检测，使得开发团队能够快速地修补漏洞。

静态分析工具的使用和适合环境的供应商的选择是成功的另一个关键技术要素。任何有利于自动化软件开发过程中任何部分的技术都是值得欢迎的，但因为并没有在选择工具或工具集的过程中使用合适的人和过程，这个软件已经成为许多组织的搁置软件（shelf-ware）。在这个空间中不是所有的工具都是一样的，在某些语言上有些工具表现得更好，尽管其他工具有很大的治理 / 风险 / 合规性（Governance/Risk/Compliance, GRC）和度量分析。在某些情况下，需要使用多达三个不同的工具才能有效实施。最后，需要选择合适的工具，需要支持语言、可伸缩、可以嵌入到开发过程中以及拥有最低的误报率。

软件开发是一项复杂的业务，对于大多数组织来说，任何能够使这一过程更加可重复、可预测并减少“摩擦”的举措都是一个巨大的胜利。使用静态分析工具有很多好处。最重要的原因有以下几个。

- 静态分析工具可以缩短时间。它们可以非常快速地评审大量的代码，这是人类无法做到的。
- 静态分析工具不会感到疲惫。静态分析工具在凌晨 2:00 连续运行 4 个小时和在上班时运行效果不变。但对于人类评审者来说，效果是不可能一样的。
- 静态分析工具帮助开发人员了解安全漏洞。在许多情况下，可以使用这些工具和来自供应商的教育资源，对开发团队开展软件安全方面的教育。

2.5.1.3 动态分析

动态程序分析是指在真实或者虚拟处理器中运行程序的条件下，进行计算机软件分析的方法。目的是在程序运行时找到程序的安全错误，而不是反复地离线审查代码。通过在设计程序的所有场景下调试它，动态分析避免了人为创造可能产生错误的情景。它有一个显著的优势，即有能力识别可能已经漏报的漏洞，以及验证静态代码分析的结果。

动态分析，也称为动态应用安全测试（Dynamic Application Security Testing, DAST），能够确定应用程序中的漏洞。这些工具用于快速地评估系统的整体安全，并使用在 SDL 和 SDLC 中。有关使用静态分析工具的优点和注意事项同样适用于动态分析工具。一些流行的 SAST 供应商的产品包括：Coverity^[38]、HP Fortify Static Code Analyzer^[39]、IBM Security AppScan Source^[40]、Klocwork^[41]、Parasoft^[42] 和 Veracode^[43]，而比较流行的 DAST 供应商的产品包括：HP WebInspect^[44]、QAINspect^[45]、IBM Security AppScan Enterprise^[46]、Veracode^[47] 和 WhiteHat Sentinel Source^[48]。

SDL 中使用 SAST 和 DAST 工具的时机是至关重要的，并且主要发生在 SDL 的设计和开发阶段，这将会在本书后面章节介绍，如图 2-1 所示。

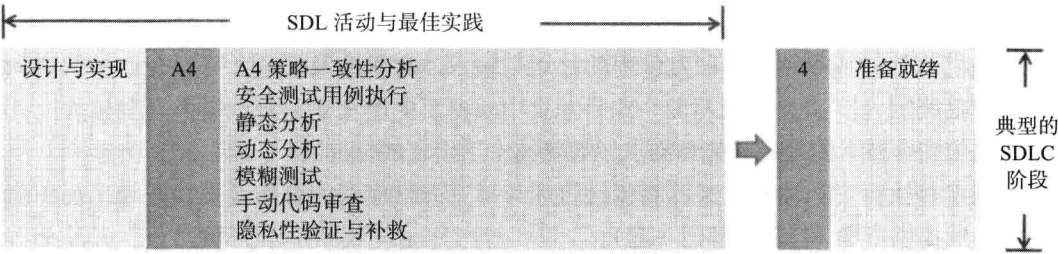


图 2-1 SDL 活动和最佳实践中的设计和开发（A4）阶段

2.5.2 人才

2.5.2.1 软件安全架构师

正如第 1 章提到的，合格的高级软件架构师将会创造或者破坏软件安全计划。在人们面前，一个有效的软件安全计划最关键的要素是级别为 3 和 4 的高级软件安全架构师。在进入安全领域之前，他们都拥有 5 ~ 10 年的开发 / 编码经验，还有软件、网络和 Cloud/SaaS 架构设计领域的经验。

这些都不是 IT 安全中运行工具的典型人员。他们是熟悉开发、系统架构和安全的资深架构师。此外，他们还必须有很好的政治和人际交往能力。他们要接触 SDLC 和 SDL 中的每个要素，他们应该是 SDLC 过程每个阶段审批过程的一部分，并且他们必须参与从预提交至后

期发布阶段的工作。他们将会创造或者打破软件安全实践，并且是其生存和成功的关键。高级软件 / 应用程序安全架构师在以下几个方面是至关重要的：处理产品安全升级、培训开发团队成员、提供内部 / 外部客户的反应以及解决复杂的 SaaS 和云环境下的软件 / 应用程序问题。

区分架构驱动和其他需求并不简单，因为它需要全面了解解决方案的目标。软件安全架构是一个一个交互式过程，它涉及系统需求业务价值的评估和确定系统成功最关键的需求。这些需求包括：功能需求、约束和解决方案的行为特性，所有这些都必须分类并且指定。因为这些关键需求塑造了系统的设计，它们称为架构驱动。

安全架构师必须弄清楚，在架构层面，如何将必要的安全技术集成到整个系统中。在云计算或 SaaS 环境中，这包括网络安全需求，如防火墙、虚拟专用网络等。架构师通常通过绘制信任边界（例如，防火墙之外的网络流量是不可信的，但本地流量是可信的），明确记录系统每一部分的信任假设。当然，这些边界必须反映业务需求。例如，高安全性的应用程序应该不愿意相信任何未加密的共享网络介质。安全需求应该来自用户。一位经验丰富的软件安全架构师典型的工作描述可能是这样的。

软件安全架构师负责提供整个 X 公司产品安全的架构和技术指导。产品安全架构师将设计、计划和实现安全编码实践和安全测试方法；确保该实践符合软件认证过程；推动产品的安全测试；测试和评估安全相关的工具；管理第三方供应商以满足上述这些责任。具体的角色和职责包括以下方面。

- 推动整体软件安全架构。
- 提供技术指导，如在全面计划、开发和 X 公司软件安全努力的执行中提供技术指导。
- 与产品和工程开发团队紧密合作，确保产品符合或超过用户的安全和认证需求。其中包括确保安全架构具有良好的文档记录并经过沟通。
- 提供给软件工程和产品开发过程的计划与输入。这里涉及安全，并对企业的约束和需求很敏感。
- 监控安全技术的发展趋势和需求，如新技术下出现的标准。
- 制定并执行安全计划。这可能包括管理与第三方供应商的合作开发，并为工程和测试实践提供指导（与其他部门一起）。
- 保证和创建（如果需要）安全策略、过程、实践和操作，确保可重复开发和高质量，同时控制成本。
- 从事实际深入的软件分析、评审和设计，包括从安全角度来看源代码的技术评审和分析。这将会包括内部开发代码和第三方供应商提供的技术评审。
- 在安全认证过程中提供主要技术角色，包括准备大量的文档和与第三方评估合作。
- 为员工、合约商、开发人员、质量保障团队和与产品安全相关的产品 / 软件安全拥护者提供培训。
- 指导 X 公司软件开发团队通过面向 SDLC 的安全开发生命周期（SDL）设计，参与设计评审、威胁建模、代码和系统的深入安全渗透测试。把这些职责延伸到提供应用设计的输入、安全编码实践、日志取证、日志设计和应用代码。

软件安全架构师在监督和培训软件安全拥护者方面是至关重要的，软件安全拥护者应该通过跨业务单元 / 软件的安全教育和认知计划来确定。架构师也将发现和评估软件安全拥护

者的候选人，从概念提交到后发布阶段，他们都参与各种软件产品的 SDL。

2.5.2.2 软件安全从业者

无论是 IT、硬件还是软件，公司安全部门经费供给在可预见的未来是很难再有所改善了，这意味着如果想要成功必须非常明智地使用资源。正如我们前面暗示的，经验丰富的软件安全架构师少之又少，在今天的市场最好的情况下你不太可能能够找到并聘用数量较多的软件安全架构师。当你看到本书使用 SDL 模型或者本章前面引用的其他模型时，你可能会问自己：考虑到安全软件和合作的开发团队将会拥有的资源，你该如何扩展到这个任务。答案是，如果你管理软件安全团队或者安全团队为你工作，你将会招聘软件安全从业者（Software Security Champion, SSC）来管理这项艰巨的任务。该角色的候选者通常应该具有至少 3 ~ 5 年的软件开发经验；愿意从事软件安全工作或者有相关的背景；有时间接受软件安全和中央软件安全团队工具、计划和过程的培训；最重要的是，不仅要知道如何开发（develop/build）软件，还要了解如何“像黑客一样思考”并拆解（deconstruct/take it apart）软件，即要综合考虑攻击者的利用软件采用的所有可能途径或攻击（exploit/attack plane）。每一个产品开发组织应该至少有一个人拥有技术能力并能被训练成软件安全从业者，并最终作为初级软件安全架构师协助中央软件安全团队进行架构安全分析 / 威胁建模。同样重要的是，SSC 是自愿的，不是受托人，否则他们可能缺乏在这个非常挑战但有益的角色上成功的激情。公司内软件开发的每一个业务部门应至少有一个 SSC；对于较大的开发组织，最好是每个业务部门每一层产品都有一个 SSC。一个软件安全拥护者典型的工作描述如下所示。

- SSC 必须有至少 3 ~ 5 年的软件开发经验；愿意从事软件安全或者有相关的背景；有时间接受软件安全和中央业务单元特定的软件安全团队工具、计划和过程的培训；最重要的是，不仅要知道如何开发软件，还要了解如何“像黑客一样思考”并拆解软件，即考虑到攻击者利用软件采用的所有可能的途径或攻击。
- 每一个产品开发组织应该至少有一个人拥有技术能力并能被训练成软件安全从业者，并最终作为初级软件安全架构师协助中央软件安全团队进行架构安全分析 / 威胁建模。理想情况下，除了面向技术的产品安全从业者，如果认为有必要，每一个团队应该有一个额外的产品安全从业者协助作为一个变更代理（更多面向项目 / 计划的个人）。
- 具体的角色和职责包括以下几个。
 - 强制实施 SDL：协助中央软件安全团队保证安全租户的机密性、完整性和可用性都持续包含在 SDL 中，作为 X SDL 公司的一部分。
 - 评审：协助中央软件安全团队软件安全架构师构建架构安全分析、评审和威胁建模。
 - 工具专家：代表每一个开发团队、产品团队和 / 或业务单元中的中央软件安全团队软件安全工具专家（例如，静态、动态分析和模糊测试）。
 - 搭配：作为每一个开发团队、产品团队和业务单元的中央软件安全团队的眼睛、耳朵和支持者。
 - 参加会议：作为一个全球 X 公司软件安全领军团队的成员，参加每月的电话会议和一年两次的面对面会议（在预算允许的前提下）。

2.6 最小特权原则

在信息安全、计算机科学以及其他领域，最小特权原则（the principle of least privilege）

规定,在计算环境特定的抽象层中,每一个模块(如过程、用户或程序,取决于主题)仅仅能访问合法目的所必需的信息和资源。^[49,50]

限制特权提升是威胁建模的重要部分,其也是 SDL 架构(A2)阶段的一个核心组成部分,这将在第4章讨论。特权提升的概念是非常重要的,它是微软安全开发生命周期卡片游戏的主题,旨在培养开发人员和安全专业人员快速、轻松地找到软件或计算机系统的威胁。^[51]未经授权的权限提升攻击利用程序错误或设计缺陷,赋予攻击者访问网络及其相关的数据和应用程序的提升权限。这些攻击可以是垂直的(攻击者赋予自己特权),或水平的(攻击者使用已经赋予的相同水平的特权),但假设他拥有具有相似权限的另一个用户的身份。

确保最小特权防止敏感数据泄露,并防止未经授权的用户访问程序或者他们未曾想要访问的区域。软件设计应遵循最小特权原则,这是软件发展的一个关键因素。限制特权等级是非常重要的,因为特权提升可以导致攻击者获得超过授予一个普通用户的授权。例如,若拥有普通用户特权的攻击者有“只读”权限,则他或许能够破解软件进而提升权限以获得“读写”权限。推动最小权限要求用户获得执行一个特定任务所必需的最小权限。在 SDL/SDLC 的设计阶段,你需要确定执行工作所需的特权最小集合,限制用户登录到只有这些特权的域中。确保最小特权不仅包括用户权限的限制,还包括资源权限的限制,比如 CPU 限制、内存、网络和文件系统权限。这就要求将权限授予一个对象之前满足多个条件,因为仅仅检查一种状况下的访问可能无法满足强安全性。例如,如果攻击者能够获得一个特权,但没有获得第二个特权,将会限制其实施成功的攻击。将软件隔离到需要多次检查访问的单独组件中,可以抑制攻击或者可能阻止攻击者接管整个系统。访问权限的慎重授权可以限制攻击者成功地攻击软件或系统。资源的最小权利和访问应限于完成该任务需要的最短时间内。

2.7 隐私

保护用户的隐私是 SDL 过程的另一个重要组成部分,应被视为 SDLC 所有阶段重要的系统设计原则。用户隐私安全出了问题将导致信任危机。由于在未经授权访问的用户越来越多的情况下,个人信息在媒体中公开,在软件和系统中使得客户的数据得到可靠保护的状况日益恶化。此外,许多新的隐私法律法规强调了在软件 and 系统的设计和开的包含隐私的重要性。至于安全性,已经通过的软件开发生命周期的进度的变更是成本很高的,就是把高成本的隐私保护方法和技术融入 SDLC 的适当阶段,以维护个人的隐私,并保护个人身份信息(PII)的数据。包含在微软 SDL 中的一些关键的隐私设计原则,包括提供有关数据收集、存储或者共享的适当通知,使用户可以对自己的个人信息做出明智的决策;使用户策略和控制可用;最小化数据收集和敏感性;以及存储保护和数据传送。^[52]

至关重要,隐私保护措施通过 SDL 实现的最佳实践构建到了 SDLC 中。忽略用户的隐私问题可能会导致诉讼、媒体负面报道和不信任。我们已经将隐私保护最佳实践方案部署到 SDL 中,这将在以后的章节中详细叙述。

2.8 度量标准的重要性

Lord Kelvin 曾说过,“如果你不能衡量它,就不能改进它。”^[53]这句格言今天也是如此,因为它适用于产品的安全性和满足软件开发组织的安全状况精准度量的需求。有意义的安全指标是至关重要的,因为企业要合作努力应对监管和风险管理要求,用敏锐的证券投资来加

强安全预算，把客户要求证明的安全性和私密性内置到他们的产品中，而不是通过发布后的修复。

指标跟踪就像是软件项目的一个保险，也协助管理漏洞防范。正如我们已经多次指出，与在 SDLC 起源阶段检测出相同的缺陷相比，在 SDLC 的连续阶段检测该缺陷的成本是非常高的。在整个 SDL/SDLC 过程中指标可以跟踪这些成本，并对 ROI（回报率投资）计算提供显著帮助。如图 1-1 所示，在开发早期避免潜在的安全缺陷花费比较少，尤其是相对于开发后期成本 10%、20%、50% 甚至 100% 的增长。在 SDLC 的不同阶段作为 SDL 过程的一部分修复缺陷成本的可视表示见图 2-2。可以说，防止一个或两个缺陷出现对于跟踪的指标的成本是物有所值的。预见的缺陷和纠正他们的能力是一个健康的软件安全计划的一个很好的指标，但质量指标在整个 SDL/SDLC 过程中可以在管理和避免过高补救成本时有所帮助。

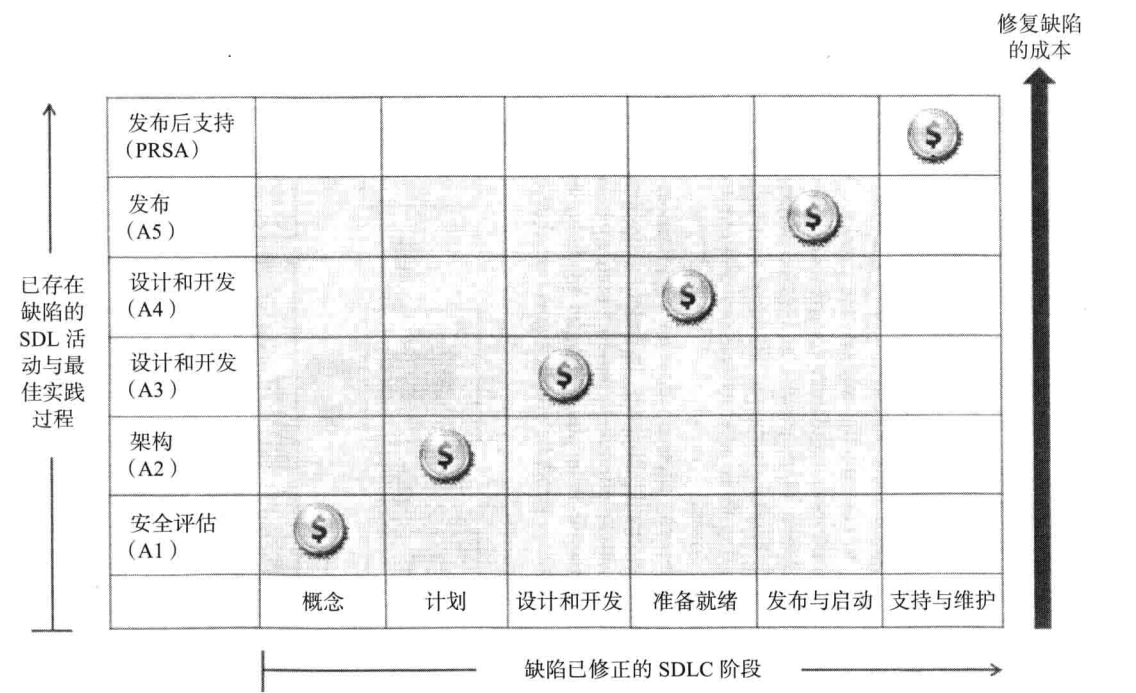


图 2-2 在 SDLC 不同阶段（SDL 过程的一部分）修复缺陷成本的可视化表示

SDL 的一个目标是在整个过程中实现缺陷多级过滤过程，而不是仅仅包含单个活动或时间节点，从而使导致漏洞的剩余缺陷最小化。每个缺陷去除活动可以被看作一个从软件产品中消除一定百分比缺陷的过滤器，这些缺陷可导致安全漏洞。^[54] 在软件开发生命周期中有越多的缺陷去除过滤器，当软件产品发布时，其中剩余的可导致安全漏洞的缺陷将越少。更重要的是，早期的测量使组织能够于软件开发生命周期的早期采取纠正措施。每一次缺陷去掉时，都会测量它们。每个缺陷移除的点都会成为一个检测的点。缺陷度量比缺陷的移除和预防更重要：它告诉团队他们的立场与目标，帮助他们决定是否要转移到下一个步骤，或停止并采取纠正措施，并指示如何改正以满足他们的目标^[55]。本书将重点介绍 SDL 模型中如何在 SDLC 中筛选出缺陷，模型的重要阶段 S1-S3 见图 2-3。

安全度量标准是可用于评估一个组织软件安全计划有效性的宝贵资源。有意义的度量标

准，可用于不断改进产品安全程序的性能，证明合规性，提高管理和利益相关者的安全意识水平，并协助决策者提供资金的请求。如果没有度量标准，每个企业都恐惧、不确定和怀疑下管理其产品。

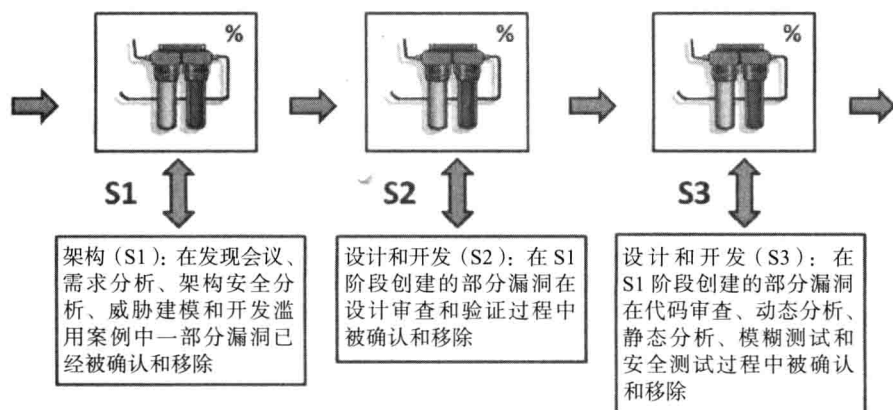


图 2-3 SDL 阶段 S1 ~ S3：缺陷识别和修复过滤过程

有意义的安全度量标准允许一个组织确定其安全控制的有效性。为了有效地衡量一个组织的安全状况，产品的安全性首先要保证适当的框架到位，以得出有意义的度量数据。这包括适合于产品的安全治理模式、企业的战略和运营要求。这样的模式应该实现实用的产品安全政策和程序，统一部署最佳做法和措施，并要求在整个组织强有力地执行管理支持。最佳实践要求下的安全性是作为一个企业的水平、垂直和跨职能管理整个组织的模型。这种模式更适合于实现一致的监测、计量和报告企业的产品安全状况。

对有效测量的安全性，它必须得到有效的管理。当公司努力保护宝贵的信息资产和论证基于风险的决策时，集中度量报告机制对于产生有意义的指标，并提供软件开发组织中的产品安全状况的持续评估至关重要。

我们的方法将包含在 SDL 模型中给出的每一个步骤的指标，而不是包含在单独的章节中。这将累积在管理整个企业的软件安全程序中使用 SDL 指标的讨论。

2.9 把 SDL 映射到软件开发生命周期

无论你使用何种 SDL 模式，不论它是否存在，一个你自己开发的或者是两者的结合，都必须将其映射到当前的 SDLC 才能有效。图 2-4 是一个 SDL 活动和最佳实践模型，作者已经开发并将其映射到典型的 SDLC 阶段。每一个 SDL 活动和最佳实践基于现实世界的经验，作者提供的例子向读者表明安全可以内置到每一个 SDLC 阶段中——安全到 SDLC 的映射。如果安全内置到每一个 SDLC 阶段，那么该软件默认有较高的概率是安全的，后续软件的变化是不太可能危及到整体安全的。这种映射的另一个好处是，你将有可能与 SDL 的拥有者和利益相关者一起工作，这将有助于在 SDLC 的操作和业务过程中构建买进、高效和可实现的安全，将可能包括开发人员、产品和项目经理、业务经理以及主管。

后续章节将详细介绍 SDL 的每一个阶段，图 2-4 详细列出了每一个阶段，图 2-5 ~ 图 2-10 分别说明。

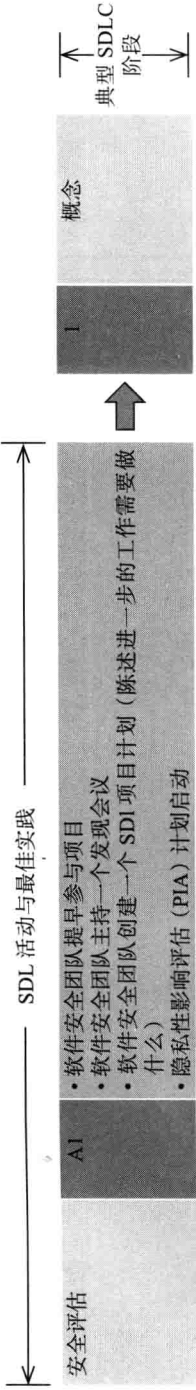


图 2-5 第 3 章

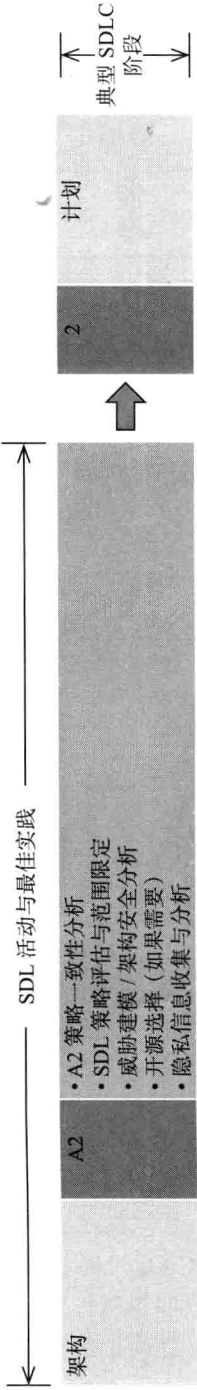


图 2-6 第 4 章

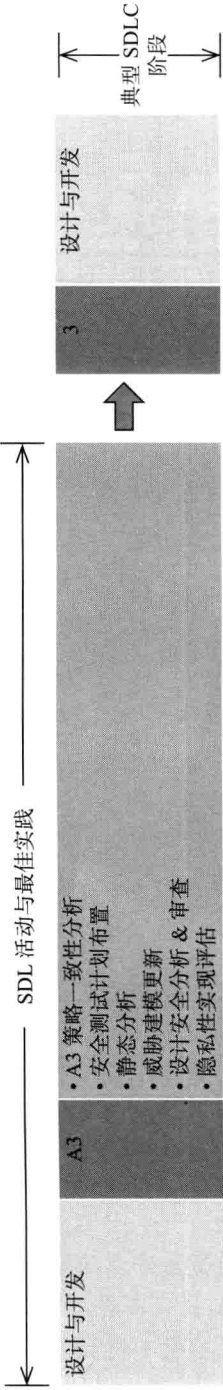


图 2-7 第 5 章

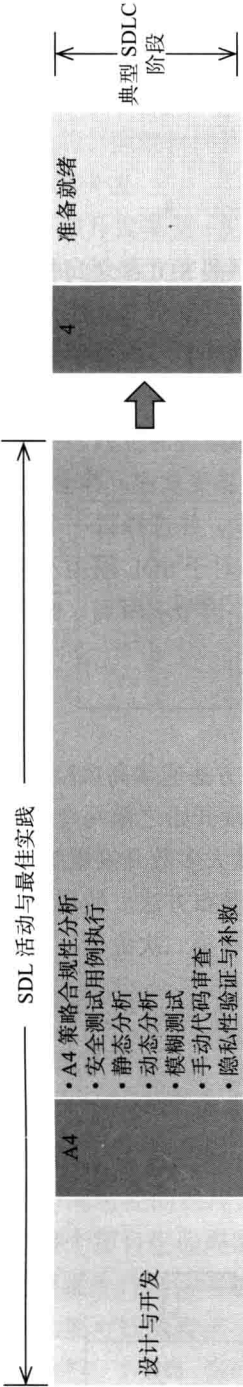


图 2-8 第 6 章

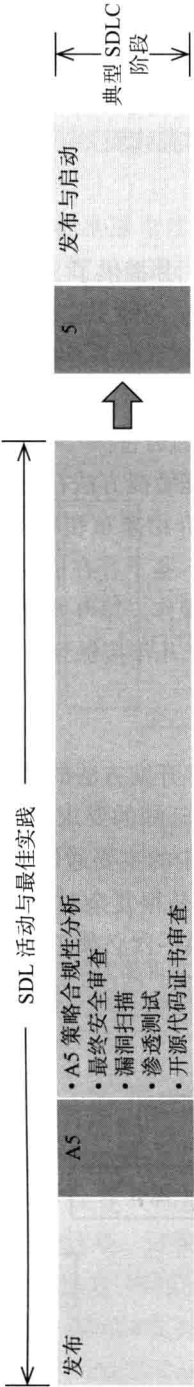


图 2-9 第 7 章

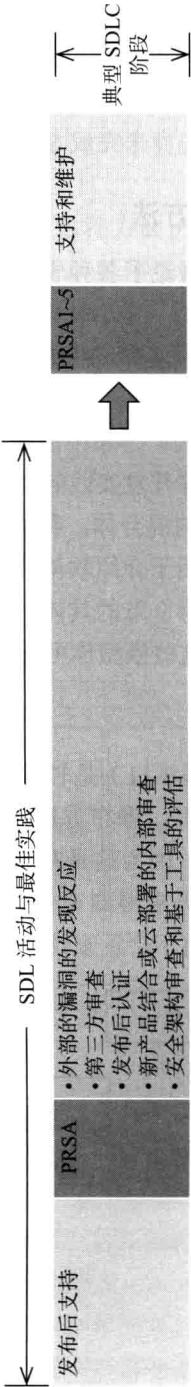


图 2-10 第 8 章

请注意，不同于已经见过的一些 SDL，我们把发布后的支持活动和最佳实践列入 SDL 中，如图 2-10 所示。因为大多数软件安全团队或类似机构，特别是那些中小型公司，没有独立的产品安全事件响应小组（Product Security Incident Response Team, PSIRT），该团队专门致力于执行安全 M&A 评估、第三方评审、发布后认证、面向云部署的新产品组合的内部评审或者面向仍在使用或即将重用的传统软件的评审。这需要一些盒子外面的思维与一个小团队管理这一切。本书后面的章节将讨论利用经验丰富的软件安全架构师、软件安全拥护者、专业软件和第三方承包商来完成 SDL 的目标与活动。

2.10 软件开发方法

本章前面已经讨论了各种 SDLC 模型，并提供了 SDL 模型和一般 SDLC 之间映射关系的直观概述。应当指出，多个软件开发方法在各种 SDLC 模型中使用。每一个软件开发方法都作为应用特定框架开发和维护软件的基础，不太关心技术方面，相反关心创建软件过程的组织方面。这些开发方法主要是瀑布模型、敏捷及其变种和衍生模型。瀑布模型是最古老和最知名的软件开发方法。瀑布模型的显著特点是它从需求按顺序循序渐进的过程。尽管它们包括传统和新型的软件开发实践的混合，但是敏捷方法在行业中越来越受欢迎。你会发现敏捷、传统瀑布或者两者的混合体。我们将详细介绍瀑布和敏捷开发模式，并选择每一个模式对应的一个或两个变体用于介绍软件开发方法。鉴于存在的模型数量，对于 SDL 模型不仅有一个通用模型，还将按第 9 章的具体介绍进行操作。第 9 章描述了 SDL 模型应用到一些最流行的软件开发模型中，这些模型你可能会在未来几年接触到。

2.10.1 瀑布开发

瀑布开发（见图 2-11）是较传统的软件开发方法的别称。这种方法通常高风险、高成本，而且比敏捷方法低效。瀑布方法使用那些已知的要求，下一个阶段开始之前每个阶段结束，并需要大量的文档，因为这是在整个过程中的主要通信机制。虽然大多数开发组织正在转向敏捷开发方法，当需求得到充分理解或不是很复杂时，依然使用瀑布方法。使用瀑布方法，一旦某一阶段完成将不会重复访问，因此第一次必须正确：通常没有第二次机会。

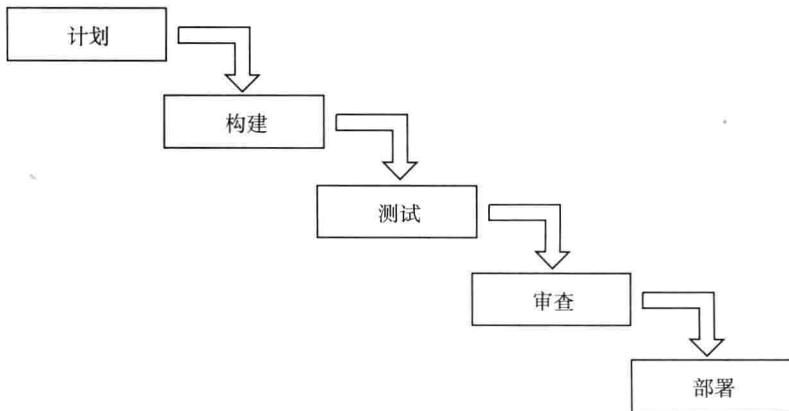


图 2-11 瀑布软件开发方法

虽然瀑布开发方法有所不同，但它们往往是类似的，即参与者尽量保持最初的计划、直

到周期的晚期才有可行软件，假设他们知道前期的所有事情，通过一个变更控制委员会最小化变化（即，假设变化是坏的和可以控制的），给予项目经理（Project Manager, PM）大部分责任，优化与进度和预算的一致性，通常采用弱控制，并且允许仅在完成时实现价值。它们由一个以 PM 为中心的方法驱动，即如果遵守计划中的过程，那么一切工作将会按照计划进行。在今天的开发环境中，在前面的句子中列出的大部分项目被认为是瀑布方法的负面属性，这是行业转向敏捷开发方法的原因。瀑布方法可以看作一个流水线方法，当恰当地应用于硬件时它是出色的，但就软件开发而论，它与敏捷方法相比有缺点。

迭代瀑布开发

迭代瀑布开发模型（见图 2-12）是标准瀑布模型的改进。这种方法比传统的瀑布方法风险较小，但比敏捷方法风险大且效率较低。在迭代瀑布方法中，整体项目分为不同的阶段，每一阶段采用传统的瀑布方法各自执行。分解较大的项目为较小的可识别的阶段，将会导致每一个阶段对应一个较小的范围，在转向下一个阶段之前，根据需要，每一个阶段的最终交付成果能够被评审和改进。因此，整体风险降低。虽然迭代法是传统瀑布方法的改进，但是在今天的环境中，你更可能要面对的是敏捷软件开发方法，而不是标准或者迭代瀑布方法。

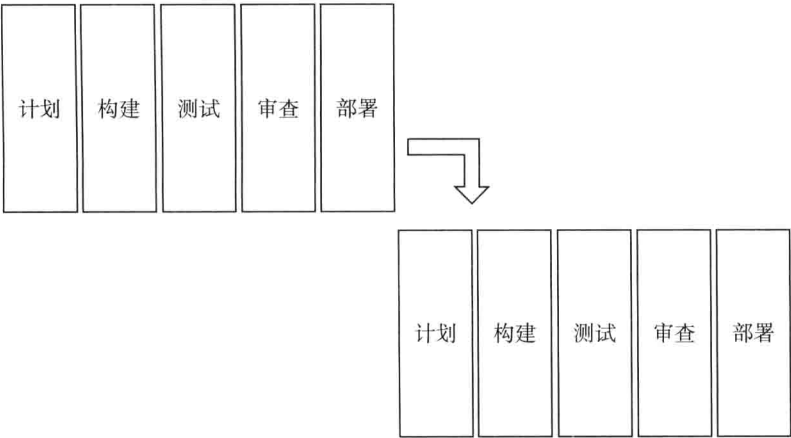


图 2-12 迭代瀑布软件开发方法

2.10.2 敏捷开发

敏捷方法是基于迭代和增量的开发方法。需求和解决方案通过自组织、跨职能的团队之间的协同推进，并且从每一次迭代中产生的解决方案在整个过程中定期回顾和提炼。敏捷方法是一个时间可控的迭代方法，有利于快速灵活地响应变化，从而鼓励演进式开发并交付，同时促进整个项目生命周期中的适应性计划、开发、团队合作、协作和过程适应性。任务被分解成需要最小计划的增量。这些迭代称为“时间盒”，它可以持续 1 ~ 4 周。发布一个产品或多个新功能可能需要多次迭代。跨职能团队负责每次迭代中所有的软件开发功能，包括计划、需求分析、设计、编码、单元测试和验收测试。一个敏捷项目通常是跨职能的，自组织团队与任何层次的企业或个人团队成员（自己决定如何满足每一个迭代的需求）独立工作。这使得项目能够快速适应变化并最小化整体风险。目标是在每一次迭代的结束，有一个可用的发布和一个展示给利益相关者的可行产品。

2.10.2.1 Scrum

Scrum (见图 2-13) 是一种迭代和增量敏捷软件开发方法, 用于管理软件项目、产品或者应用程序开发。Scrum 采用一种经验方法, 接受该问题不能被完全理解或定义, 而将重点放在最大限度地提高团队的交付, 并迅速响应新出现的需求。这是通过使用协同定位和所有学科可以表示的自组织团队完成的。相较于传统计划或预测的方法, 这个概念有利于促进客户 (在项目开发过程中改变需求) 处理生产的能力。Scrum 开发的基本单元称为一个“冲刺”(sprint), 能够持续一周到一个月。每个冲刺是时间可控的, 使产品的成品部分按时完成。这些需求的优先级列表是从产品积存中衍生的, 如果它们在冲刺过程中没有完成, 它们将被忽略并返回产品积存中。该团队演示了每一个冲刺完成后的软件。Scrum 普遍接受的增值属性包括自适应计划的使用; 它需要第一个冲刺 (通常为两周) 早期可行软件的反馈; 它强调好的改变的最大化, 如专注于最大限度地提高整个项目的学习; 它把大部分责任放在小的、专注于思维的适应性团队, 计划和重新计划自己的工作; 它具有较强的和频繁的控制; 优化业务价值、上市时间和质量; 支持较早实现价值 (可能在每次冲刺后)。

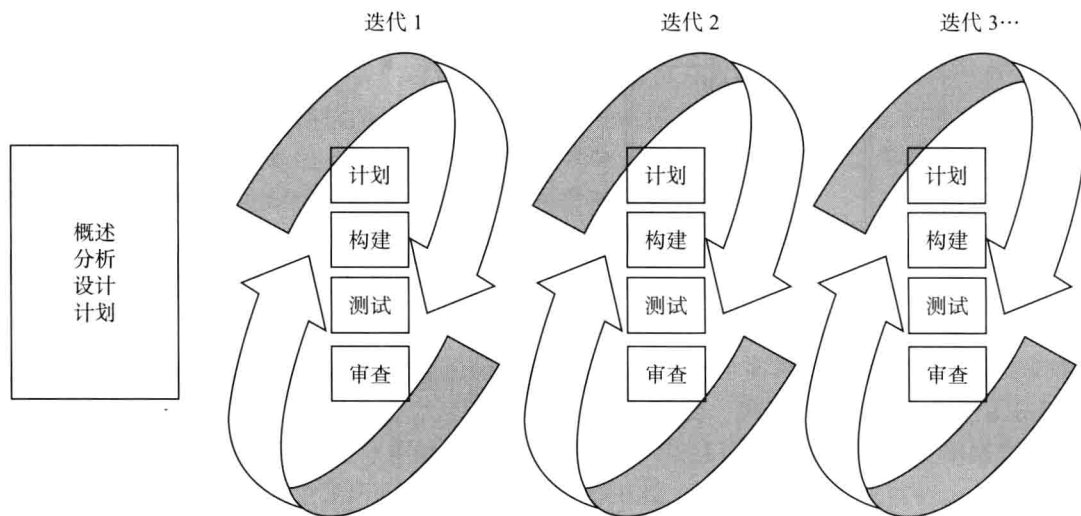


图 2-13 Scrum 软件开发方法

2.10.2.2 精益开发

根据我们的经验, 对于那些已经或者正在从瀑布方法中转变的软件开发者来说, Scrum 是敏捷开发各种方法中最有可能遇到的。精益 (lean, 见图 2-14) 是另一种日益普及的方法, 值得一说。遗憾的是, 精益有许多定义, 它是一种在很多方向发展的方法。虽然和 Scrum 类似, 精益也关注功能而不是群功能, 但精益将这个想法更进一步, 即在选择、计划、开发、测试和部署下一个功能之前, 以最简单的形式选择、计划、开发、测试和部署当前功能。其目的是进一步隔离个体功能等级的风险。这种隔离的优点是专注于尽可能避免“浪费”, 除非绝对必要或者相关, 否则将什么都不做。基于精益生产原则的概念, 精益开发方法可以概括为 7 个基本“精益”原则: (1) 避免浪费, (2) 增强学习, (3) 尽量推迟决策, (4) 尽快交付, (5) 授权团队, (6) 嵌入完整性, (7) 着眼整体。精益开发方法的关键要素之一是提供一种着眼整体的模型, 即使开发人员分散在多个位置和合约商中。虽然社区中许多人依然认

为其和敏捷相关，但精益软件开发方法已经演变成一门相关学科，而不是一个特定的敏捷开发方法。



图 2-14 精益软件开发方法

2.11 本章小结

本章描述了 SDL 的重要性和适用性，以及与 SDLC 的关系。在整个讨论过程中，我们强调了模型、方法、工具、人才和度量标准，以管理和克服实现软件安全的挑战。SDL 过程包含 SDL 每一个阶段的一系列以安全为中心的活动与最佳实践。这些活动和最佳实践包括软件设计过程中威胁模型的开发，实现过程中静态分析代码扫描工具的使用，以及代码审查、安全测试和度量标准的管理。最后，我们讨论了把 SDL 映射到 SDLC 和各种流行的软件方法，在第 9 章中把 SDL 的要素和最佳实践应用于这些软件方法。下一章将开始介绍 SDL 模型的第一步，并展示 SDL 要素的增量实现，将增量改进整体的软件安全性。

参考文献

1. Microsoft Corporation (2012), Graphic for Microsoft SDL. Retrieved from <http://www.microsoft.com/security/sdl/discover/default.aspx>.
2. Addison-Wesley (2012), Software Security Series. Graphic for “Build Security in for Seven Touchpoints for Software Security.” Retrieved from <http://www.buildsecurityin.com/concepts/touchpoints>.
3. OWASP (2012), OWASP: The Open Web Application Security Project—Security Code Review in the SDLC, Secure Code Review Process—Operational Process. Retrieved from https://www.owasp.org/index.php/Security_Code_Review_in_the_SDLC.
4. Cisco Systems (2012), Cisco Secure Development Lifecycle (CSDL) Graphics. Retrieved from <http://www.cisco.com/web/about/security/cspo/cSDL/index.html>.
5. Microsoft Corporation (2012). Microsoft Security Development Lifecycle: The SDL Optimization Model Graphic. Retrieved from <http://www.microsoft.com/security/sdl/learn/assess.aspx>.
6. Microsoft Corporation (2012), “Microsoft SDL Optimization Model.” Retrieved from <http://www.microsoft.com/download/en/details.aspx?displaylang=en&id=2830>.
7. Bsimmm.com (2012), Building Security in Maturity Model.pdf. Graphic for the BSIMM Software Security Framework (SSF), p. 24. Retrieved from bsimm.com/download/dl.php.
8. OWASP (2012), “Software Assurance Maturity Model (SAMM).” Retrieved from https://www.owasp.org/index.php/Software_Assurance_Maturity_

Model_(SAMM).

9. Businesswire.com (2012), “BSIMM4 Release Expands Software Security Measurement Tool and Describes New Activities.” Retrieved from <http://www.businesswire.com/news/home/20120918005298/en/BSIMM4-Release-Expands-Software-Security-Measurement-Tool>.
10. Ibid.
11. OWASP (2012), “Software Assurance Maturity Model (SAMM).” Retrieved from [https://www.owasp.org/index.php/Software_Assurance_Maturity_Model_\(SAMM\)](https://www.owasp.org/index.php/Software_Assurance_Maturity_Model_(SAMM)).
12. ISO (2013), “ISO/IEC 27034-1:201: Information Technology—Security Techniques—Application Security—Part 1: Overview and Concepts.” Retrieved from http://www.iso.org/iso/catalogue_detail.htm?csnumber=44378.
13. Pickel, J. (May 2013), “ISO/IEC 27034—Why, What, and How.” PowerPoint presentation at the 2013 Microsoft Software Development Conference, delivered on February 25, 2013, San Francisco, CA.
14. Ashord, W. (May 13, 2013), “Microsoft Declares Conformance with ISO 27034.” Computer Weekly.Com. Retrieved from <http://www.computerweekly.com/news/2240184149/Microsoft-declares-conformance-with-ISO-27034-1>.
15. SAFECode (2012), SAFECode “About Us” webpage. Retrieved from http://www.safecode.org/about_us.php.
16. SAFECode (2011), *Fundamental Practices for Secure Software Development, 2nd ed., A Guide to the Most Effective Secure Development Practices in Use Today*, February 8, 2011. Retrieved from www.safecode.org/publications/SAFECode_Dev_Practices0211.pdf.
17. U.S. Department of Homeland Security (2012), “Build Security In.” Retrieved from <https://buildsecurityin.us-cert.gov/bsi/home.html>.
18. U.S. Department of Homeland Security (2012), “Background: Department of Homeland Security (DHS) National Cyber Security Division’s (NCSA).” Retrieved from <https://buildsecurityin.us-cert.gov/swa/cwe/background.html>.
19. U.S. Department of Homeland Security (2012), “Software Assurance: Community Resources and Information Clearinghouse.” Retrieved from <https://buildsecurityin.us-cert.gov/swa/cwe>.
20. U.S. National Institute of Standards and Technology (2012), “Introduction to SAMATE.” Retrieved from http://samate.nist.gov/index.php/Introduction_to_SAMATE.html.
21. U.S. National Institute of Standards and Technology (2008), NIST Special Publication 800-64, Revision 2: *Security Considerations in the System Development Life Cycle*, October 2008. Retrieved from <http://csrc.nist.gov/publications/nistpubs/800-64-Rev2/SP800-64-Revision2.pdf>.
22. U.S. National Institute of Standards and Technology (2012), *National Vulnerability Database*, Version 2.2. Retrieved from <http://nvd.nist.gov>.
23. U.S. National Institute of Standards and Technology (2012), “NVD Common Vulnerability Scoring System Support v2.” Retrieved from <http://nvd.nist.gov/cvss.cfm?version=2>.
24. MITRE Corporation (2012), Common Vulnerabilities and Exposures (CVE) homepage. Retrieved from <http://cve.mitre.org>.
25. MITRE Corporation (2012), “CVE Frequently Asked Questions.” Retrieved from <http://cve.mitre.org/about/faqs.html>.
26. SANS Institute (2012), “Twenty Critical Security Controls for Effective Cyber Defense: Consensus Audit Guidelines.” Retrieved from <http://www.sans.org/critical-security-controls>.

27. MITRE Corporation (2012), "CVE-Compatible Products and Services." Retrieved from <http://cve.mitre.org/compatible/compatible.html>.
28. MITRE Corporation (2012), "CVE Frequently Asked Questions." Retrieved from <http://cve.mitre.org/about/faqs.html>.
29. U.S. Department of Defense Cyber Security and Information Systems Information Analysis Center (CSIAc) (2012), CSIAc webpage. Retrieved from <https://www.thecsiac.com/group/csiac>.
30. Goertzel, K., et al., for Department of Homeland Security and Department of Defense Data and Analysis Center for Software (2008), *Enhancing the Development Life Cycle to Produce Secure Software: A Reference Guidebook on Software Assurance*, Version 2, October 2008. Retrieved from https://www.thedacs.com/techs/enhanced_life_cycles.
31. Goertzel, K., et al. (2008), *Software Security Assurance: State-of-the-Art Report (SOAR)*, July 31, 2008. Retrieved from <http://iac.dtic.mil/iatac/download/security.pdf>.
32. Cert.org (2013), Carnegie Mellon cert.org webpage. Retrieved from <http://www.cert.org>.
33. SecurityFocus (2013), Bugtraq website. Retrieved from <http://www.securityfocus.com/archive/1>.
34. SecurityFocus (2013), Security website. Retrieved from <http://www.securityfocus.com>.
35. National Institute of Standards and Technology (2013), National Vulnerability Database webpage. Retrieved from <http://web.nvd.nist.gov/view/vuln/search>.
36. Codenomicon (2012), Codenomicon website. Retrieved at <http://www.codenomicon.com>.
37. Peachfuzzer.com (2012), Peach Fuzzing Platform webpage. Retrieved from <http://peachfuzzer.com/Tools>.
38. Coverity (2012), Coverity Static Analysis webpage. Retrieved from <http://www.coverity.com/products/static-analysis.html>.
39. HP (2012), HP Fortify Static Code Analyzer webpage. Retrieved from <http://www.hpenterprisesecurity.com/products/hp-fortify-software-security-center/hp-fortify-static-code-analyzer>.
40. IBM (2012), IBM Security AppScan Source webpage. Retrieved from <http://www-01.ibm.com/software/rational/products/appscan/source>.
41. Klocwork (2012), Klocwork webpage. Retrieved from http://www.klocwork.com/?utm_source=PPC-Google&utm_medium=text&utm_campaign=Search-Klocwork&_kk=klocwork&gclid=CMY0_q6svbICFUjhQgodOGwAFg.
42. Parasoft (2012), Static Analysis webpage. Retrieved from http://www.parasoft.com/jsp/capabilities/static_analysis.jsp?itemId=547.
43. Veracode (2012), Veracode webpage. Retrieved from <http://www.veracode.com>.
44. Hewlett-Packard (2012), Webinspect webpage. Retrieved from <http://www.hpenterprisesecurity.com/products/hp-fortify-software-security-center/hp-webinspect>.
45. Hewlett-Packard (2012), QAinspect webpage. Retrieved from <http://www.hpenterprisesecurity.com/products/hp-fortify-software-security-center/hp-qainspect>.
46. IBM (2012), IBM Security AppScan Enterprise webpage. Retrieved from <http://www-01.ibm.com/software/awdtools/appscan/enterprise>.
47. Veracode (2012), Veracode webpage. Retrieved from <http://www.veracode.com>.
48. White Security (2012), "How the WhiteHat Sentinel Services Fit in Software Development Lifecycle." Retrieved from (SDLC)<https://www.whitehatsec.com/>

sentinel_services/SDLC.html.

49. Denning, P. J. (December 1976), "Fault Tolerant Operating Systems." *ACM Computing Surveys*, vol. 8, no. 4, pp. 359–389. DOI:10.1145/356678.356680.
50. Saltzer, J., and Schroeder, M. (September 1975), "The Protection of Information in Computer Systems." *Proceedings of the IEEE*, vol. 63, no. 9, pp. 1278–1308. DOI:10.1109/PROC.1975.9939.
51. Microsoft Corporation (2012), "Elevation of Privilege (EOP) Card Game." Retrieved from <http://www.microsoft.com/security/sdl/adopt/eop.aspx>.
52. Microsoft Corporation (2012), *Microsoft Security Development Lifecycle (SDL), Version 3.2*. Retrieved from <http://www.microsoft.com/en-us/download/details.aspx?id=24308>.
53. Quotationsbook.com (2012), Lord Kelvin quote. Retrieved from <http://quotationsbook.com/quote/46180>.
54. U.S. Department of Homeland Security (2012), "Build Security In." *Secure Software Development Life Cycle Processes* online doc. Retrieved from <https://buildsecurityin.us-cert.gov/bsi/articles/knowledge/sdlc/326-BSI.html>.
55. Ibid.

安全评估 (A1): SDL 活动与最佳实践

本章将介绍安全开发生命周期 (SDL) 的第一个阶段。这个阶段 (A1) 称为安全评估。我们将通过叙述该阶段不同的活动来说明为什么这个阶段如此重要，然后引领读者通过该阶段了解关键成功因素、可交付成果和度量标准。

安全评估 (A1) 是 SDL 的第一个阶段 (如图 3-1 所示)。这是项目团队识别产品风险及所需的 SDL 活动的短语，在一些 SDL 中它称为发现阶段。安全里程碑和控制的一个初始项目概要是，开发并集成到开发项目的进度，当发生变化时可以有适当的规划。在这个阶段中，始终有四项原则问题应该得到解决，以确定需要什么来确保软件的安全性：

1. 软件满足客户任务的关键程度如何？
2. 软件所必需的安全目标是什么 (例如第 1 章描述的保密性、完整性和可用性 (CIA))？
3. 哪些法规和政策是适用于确定什么需要保护的？
4. 在给软件运行的环境中又可能存在什么威胁？

在最初的启动会议期间，所有的利益相关者都应该参与讨论、鉴定和得到一个关于安全隐私含义的共同理解。最初的这一系列重要的安全里程碑，包括时间框架或者标志一个安全步骤将至的发展触发器，在讨论中勾勒出令开发人员规划项目中的安全要求和相关的约束。这也提醒着项目负责人，很多正在做的决策存在安全隐患，应当随着项目持续来适当权衡。这些决策也应当包含所有安全需求的资源的鉴定，包括相关法律、条例和标准。

隐私，在过去往往作为 SDL 的一部分被忽略，在评估这个阶段也是如此。隐私影响评估 (Privacy Impact Assessment, PIA) 程序评估软件中的问题和与个人身份确认信息有关的隐私影响等级，并且将在这个阶段的发展过程中启动。

3.1 软件安全团队提早参与项目

SDLC 通常有正式的启动会议，把软件安全团队包括在内是非常重要的，以确保安全是 SDLC 的重要元素，并内置于整个过程。现场会议或网络会议给了与会者和利益相关者来大致认识和了解的重要机会。开发过程的早期就引入安全团队是启动风险辨识、规划和减轻最具成本效益的方法。早期识别与减轻安全漏洞和错误配置将降低安全控制实施和漏洞缓解的成本；提高强制安全控制所造成的潜在工程挑战的认识；以及识别共享的安全服务，重复安全策略和工具，以降低开发成本，同时通过行之有效的方法和技术，提高安全性。安全团队的早期介入将使开发人员计划的安全要求和相关的约束加入到项目中。这也提醒项目负责人，随着项目的继续许多正在做出的决定应适当权衡安全隐患。早期的规划和认识将通过适当的风险管理规划，节约成本与时间。安全性的讨论应作为项目的一部分，以确保业务决策和风险影响到整体的发展这个观点成为项目人员间坚实的理解。^[1]

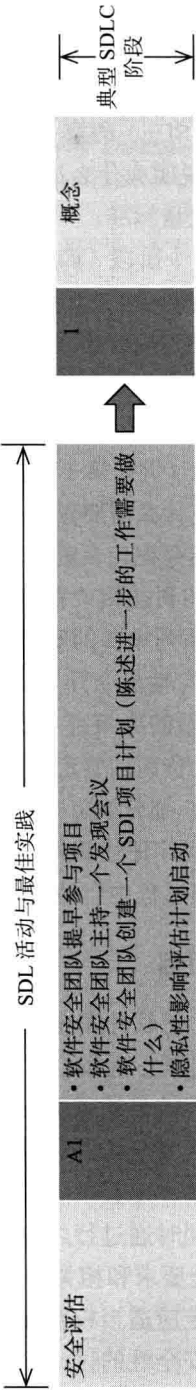


图 3-1 安全评估 (A1): SDL 活动与最佳实践

3.2 软件安全团队主持发现会议

发现会议基本上是一个 SDL 启动会议，在会上关键的 SDLC 利益相关者在过程开始时聚在一起，使安全成为内置的而不是附加的。在发现会议安全规划中应包括筹备整个系统生命周期，包括关键安全里程碑和可交付成果以及工具和技术的鉴定。特别应考虑到可能需要进行采购的项目，比如软件安全性测试和评估工具，以及第三方软件安全架构师或工程师潜在的使用可能，（如果需要扩充员工或有客户要求第三方认证）。其他资源的影响，如主动测试、认证以及所需的培训，必须加以考虑。整个系统的每一个安全注意事项应该规划在一系列里程碑或安全会议中。发现会议的成果通常是其对未来活动的决定条款，这在 SDL 过程中是后面的实际安全或隐私方面的活动。项目进度表应结合安全活动，以确保与时间表和资源相关的任何未来决定有适当的规划。所有的会议参与者和利益相关者应该就本次会议的安全影响、注意事项以及对软件的要求达成共识。

以下四个问题应在这个阶段处理，以确定被开发软件所需要的安全性控制：

1. 此系统满足组织的任务关键性有多高？
2. 软件的安全目标就机密性、完整性和可用性（CIA）来讲是什么？
3. 什么法规和政策是适用的？
4. 系统运行时在环境中有可能有什么威胁？

发现会议期间的主要任务包括以下内容。

- 制定安全性里程碑初步的项目大纲，这将被集成到开发项目的进度中，并允许规划发生适当的变化。
- 确定安全要求，如相关法律、法规、标准和客户要求的来源。
- 确定任何必要的认证和认证要求以及它们需要的资源。
- 找出任何将需要的第三方或开源软件。
- 确定用于软件开发的通用安全控件，包括那些将在 SaaS/ 云环境中使用的软件或使用多个软件产品中的更大的解决方案中被需要的控件。
- 确定战术和战略（业务）项，并定义所需的安全性报告指定条款。
- 提出关键的安全里程碑的初步框架，包括时间框架或发展触发器的安全步骤。
- 定义在 SDL/SDLC 过程中支持安全所需的核心软件安全小组、软件的安全冠军、开发商、隐私团队和任何其他利益相关者的安全责任。
- 识别和记录将被使用的软件安全设计、体系结构和安全编码实践。
- 确定使用的安全测试和评估技术。
- 制定出一个预先的隐私影响评估流程，包括确定信息的分类和识别传输、存储或创建信息（例如个人识别信息）已知的特殊处理要求，以及任何隐私要求的初步鉴定。
- 如果可能的话，项目工件，如会议纪要、简报和角色标识应标准化，并提供给开发人员最大努力的规划。这应该是整个 SDL 中一个持续的过程。

3.3 软件安全团队创建 SDL 项目计划

这实际上可以被认为是最初的项目规划，因为正式的计划将作为设计阶段的成果呈现，这将在下一章描述。在这个阶段，SDL 的项目计划应基于发现阶段获得的信息列出安全里程

碑，并把它们集成到整个 SDLC 的时间表中，使得变化发生时有适当的规划。在发现阶段，从决策方面来说，活动可能转化为安全活动遵守里程碑。这个项目计划整合了在发现阶段反映安全性和隐私活动或决策的初步时间表这样的安全预期的共识。

3.4 隐私影响评估计划启动

有许多隐私保护和管理方法。然而，在过去，隐私保护工具是普遍以自组织的方式应用的，或者是以零碎的方式解决眼前的问题；通常在这些问题解决后发布。正如安全性，在系统设计时把隐私作为次要的考虑因素，或者把它作为一个未来探讨的问题处理，并不能提供有效的隐私保护。只考虑解决隐私问题的组件，而不是通过一个全面的设计和将导致进一步的潜在隐私问题。隐私必须是一个基本的设计因素，它要集成到 SDLC 的各个阶段。

在各种层次上产生了越来越多的隐私法规，州的、联邦的、国际的，都要求对违法行为做出严厉的惩罚。我们没有用一整章来描述近期的或者即将发布的隐私要求以及它们的潜在影响，而是显示所需要的最佳实践，以充分覆盖大多数隐私、法规和政策合规性条款面对的问题。软件程序被设计为与用户的计算机集成，并因此可能能够访问和存储个人信息。软件开发人员必须遵守这一涉及操作系统和它的软件设计平台的指导方针与隐私政策。底线是，当客户委托公司敏感信息时，每个员工有义务保护这些信息。至于安全性，信任的客户的隐私被侵犯有重大的影响，这反过来会显著影响公司的声誉和你开发的软件的潜在收入。

在你开始开发一个隐私影响评估（Privacy Impact Assessment, PIA）前，你将需要评估哪些监管法规或政策适用于你正在开发的软件。在某些机型上，这就是所谓的数据敏感性的评估。由于大多数开发商没有相应的法律依据，监管机构一般没有软件开发背景，使得理解法规等问题很艰难和令人烦恼。对于开发者来说，详细了解立法语言和所描述的要求往往是非常困难的，而且这些往往是不容易明确的软件需求。为了成功地把法规转化成符合要求的，有必要聘请企业法律顾问和任何外部法律隐私专家。如果你碰巧有一个首席隐私官（CPO），这个人可能为理想合作伙伴，他为你提供的资源和培训有助于应对将隐私内置到 SDL 和最终 SDLC 中的挑战的。

微软的《开发软件产品和服务隐私准则》^[2] 和 NIST 特别出版物 800-64 修订版 2：《系统开发生命周期安全性考虑》^[3] 是在开发在 SDL 中一个 PIA 的主流参考。你可以使用全部或使用一个模板开发自己的 PIA。不管你用什么方法，以下要点都应该包含在你的 PIA 中。

- **立法摘要：**从一个开发者的角度解释行为，告诉你什么是你需要知道的，以了解其对你的应用程序开发的影响。
- **所需的工艺步骤：**需要向相关的软件开发人员解释更深入的要求。一般来说，这部分将介绍被视为敏感的数据类型，以及它们需要如何得到保护。
- **技术和技巧：**为符合法律要求解释策略和技术。这些分为五大类：机密性、完整性、可用性、审核和日志记录，以及身份验证。
- **其他资源：**提供一些链接，在那里你可以收集有关立法的更多信息。^[4]

PIA 过程的主要任务是确定系统要求以及待解决问题的初始定义。在这个阶段产生的 PIA 只是最初的系统规格和要求的一个初步版本，旨在引导开发者通过开发的早期阶段评估隐私。为简单起见，我们只包含了隐私的设计原则需求分析和初步 PIA 分析的一部分。在其核心位置，PIA 的这一阶段是规划和归档，以及对软件使用的个人身份信息（PII）和个人信息与包

含或访问以下内容评估初步需求。

- **利益相关者的教育。**所有利益相关者应在安全评估 (A1) 发现和启动会议中进行隐私设计的 4C 教育 (理解、意识、控制和同意)。架构师和开发人员应该问他们是否需要收集数据, 是否有有效的商业需要这样做, 以及客户是否支持该软件的经营宗旨, 从而为其收集 PII。
- **其他软件的交互。**外部系统流程, 与新软件交互的其他系统及其使用的 PII, 个人信息和系统的用户交互。
- **PII 收集。**PII 收集的目的和要求。
- **PII 存储保留。**提出了个人信息保留期限及理由。
- **访问。**确定哪些实体可以访问 PII 和个人信息, 以及软件中与责任 / 任务 / 角色 / 数据分离的初步设计。
- **隐私管理工具。**识别在软件和解决方案中管理个人信息可能需要的隐私管理工具和系统进程。如果软件将是一个基于 SaaS 或云的解决方案的组成部分, 这一点尤其重要。
- **安全保障。**将用于保护 PII 和个人信息的安全保障的设置要求。
- **数据完整性。**确定 PII 和个人信息是及时更新和准确的。
- **评估安全性和私密性要求之间是否有冲突。**如果有的话, 它们需要在开发过程的这个阶段处理和解决。此步骤包括隐私和安全保护, 使得软件到达要求。
- **应用最小特权原则。**从本质上讲, 这需要限制访问“谁需要知道”。访问用户数据应仅限于那些拥有合法的商业目的来访问数据的人。此外, 非使用者 (如管理员或数据库管理员) 应该只给予达到具体的商业目的所需要的最小量用户数据的访问权限。这必须包括能够访问数据的第三方或将其传递给谁: 他们只应允许访问履行他们的商业目的需要的特定数据。数据保护配置, 包括保存和销毁要求, 通过合同协议通常需要第三方确认。
- **网站和 Web 服务。**所有面向外部的网站必须有一个链接到每个页面上的隐私声明。这包括弹出窗口 (用于收集 PII)。只要有可能, 同样的隐私声明应使用一个域内的所有网站。
- **使用 cookie。**PII 和标识符便于跟踪, 它们可能存储在 cookie 中作为存储在用户的计算机上的小文件。它们保存特定于客户端和网站的适量数据, 并且可以通过 Web 服务器或客户端计算机访问。使用 cookie 的隐私准则应用于本地存储的文本文件, 允许一个服务器端连接到存储和检索信息, 包括 HTTP cookie (如网页的 cookie) 和 Flash cookie (例如, 闪存共享对象)。当一个会话 cookie 可以满足目的时不能用持久 cookie。持久 cookie 应该在达到商业目的最短的时间内到期。存储在持久 cookie 的 PII 必须加密。
- **IP 地址。**客户的 IP 地址总是与数据一起作为通过网络传输的通信协议的一部分发送。在写作本书时, 关于 IP 地址是否是 PII, 仍有不少争论和讨论。事实上, 隐私监管机构甚至讨论这是一个警告标志, 我们也许需要考虑在可预见的未来这些信息会落入 PII 的类别。如果匿名是必需的, 为了避免两者之间的相关性, 存储相同的 PII 与 IP 地址应该避免。如果可能的话, 则 IP 地址应该从有效负载被剥离, 通过限制数字的位数, 减少它的灵敏度。也可以在将 IP 地址转换为一个不太精确的位置后丢弃。
- **客户隐私的通知。**收集用户数据并将其传输的软件必须提供并发出通知给客户。这也

称为披露通知，且这些通知必须告知用户软件能够收集的信息类型以及信息将如何被使用。根据不同的软件类型，需要设置退出条款，可以让用户选择隐瞒某些类型的个人信息，如果他们想这样做。通知和要求同意的类型取决于所收集的用户数据的类型以及它将如何被使用。客户也必须呈现他们是否要共享此信息。所有通知必须写清晰，易于阅读。有两种类型的通知：重要的和可出现的。“重要通知”旨在抓住客户的注意力，并邀请客户来考察当前的隐私设置，详细了解他们的选择，并做出选择。“可发现的通知”是需要客户自己发现的。这可以通过在软件产品的帮助菜单中选择一个隐私声明链接，或通过查找和阅读网站的隐私声明完成。此通知通常包括将要存储的数据的类型，它将如何被使用，它将与谁共享，它是如何保护的，可用用户控件包括更新过程（如果 PII 的存储和可复用的），以及公司的联系信息。如果你正在开发将被其他公司使用的产品或者作为原始设备制造商（OEM），客户公司通常有特定的第三方软件开发者都必须包括的隐私声明。其他公司可能要求其产品使用的软件包括一份隐私声明，它告诉用户他们的信息不会被出售给其他公司或公开展示。软件开发人员必须告知在隐私政策和通知中软件用户个人信息的方法。正如本书后面讨论的，这可以通过一个有效的 SSL 证书，或者使用其它安全和加密方法来完成。在其他与隐私相关的领域，监管等方面的要求是动态的，你应该咨询你的隐私专家或律师为你的软件给出最新的指导。

- **儿童的隐私。**必须慎重考虑孩子的隐私，因为他们可能缺乏能力去辨别暴露他们的 PII 是否可能使他们处于危险的状况。随着社交软件中出现的协作和共享功能这变得尤为重要。家长控制功能通常需要被添加到产品、网站和 Web 服务中，以帮助保护儿童的隐私。必须作出特别努力，以确保家长保持对自己的孩子是否可以透露 PII 的控制权。针对儿童和收集他们的客户的年龄提供网站和 Web 服务方面有一些隐私要求。在这方面已经有许多州、地区和国际要求已经或即将出台要求。如果你拥有的软件将属于这个区域，请确保咨询你的隐私专家和企业法律顾问（或同等职位的人员）。
- **第三方。**在评估你的隐私要求时两种类型的第三方必须予以考虑。一种类型的第三方被授权代表公司，使用的资料均按照该公司的隐私政策。另一种是独立第三方，遵循自己的隐私惯例并且出于自己目的使用客户信息，这需要一个合同指定数据保护要求。这需要为客户提供是否同意软件条款的选项。如果 PII 是通过被批准代表公司的独立第三方传递的，仅需要“可发现的通知”。
- **用户控件。**用户控件使用户能够管理他们的数据、控制隐私和更改其设置。这些控件应该是直观的、容易找到的，并可以驻留在计算机上、Web 服务中，或者在移动设备上。一个 Web 页被用作 Web 服务的隐私网站。移动设备的隐私控件可以在设备本身上，或通过基于计算机的用户界面，或者通过网站链接到该设备。
- **需要在共享计算机上使用的软件隐私控制。**家庭或小型办公室 / 家庭办公室（SOHO）环境中，多个用户共享软件很常见。在这些环境中也收集或存储 PII 的软件，必须提供哪些用户有权访问数据的控制。这些控制可能包括电脑 / 文件 / 文件访问控制和文件权限或加密。控制也必须是默认设置，而不是可选项。共享文件夹必须被明确标注出来或者高亮显示。
- **协作、分享和社交软件隐私特征。**这是非常复杂和具有挑战的一个领域，因为其内容

可以在一个社区共享，在某些情况下，还可以与社区成员、共同朋友或联系人共享。支持这种类型应用程序的软件应该提供控制和通知，有助于防止无意识用户意想不到的 PII 共享。

- **安全。**显然，安全是本书的主题，也是隐私和质量的一个关键元素。安全要求将取决于收集到的用户数据的类型，以及是否需要本地存储、传输和 / 或远程存储。安全控制和测量的最终目标是防止 PII 丢失、误用、未经授权的访问、披露、变更和破坏。控制和测量不仅包括软件控制（诸如访问控制、传输和存储的加密），也包括物理安全、灾难恢复，以及审计。因为业务需要当标准保护不可行时，可能需要补偿控制，如使用 PII 作为一个独特的标识符，或者 IP 地址或电子邮件地址用于路由。
- **隐私影响评级。**隐私影响评级（P1、P2 或 P3）是用在微软 SDL 上的一次实践。它将从隐私的角度测量软件的待处理数据的敏感性。部署一个高隐私风险项目所必需步骤的提前意识，可以帮助你决定成本是否具有商业价值。隐私影响评级的一般定义如下。

- **P1 高隐私风险。**特征、产品、服务商店、PII 传输或者错误报告，通过匿名数据实时传输监视用户，更改设置或文件类型关联，或安装软件。
- **P2 中隐私风险。**影响特征和产品或服务中隐私的唯一行为是一次性的、由用户发起的；匿名数据传输（例如，用户点击一个链接并访问该网站）。
- **P3 低隐私风险。**特征、产品或服务中没有影响隐私的行为。没有匿名或个人数据传输，没有 PII 存储在机器上，没有代表用户改变设置，且没有安装软件。

由微软开发的风险评估问卷和风险评级体系，有助于在 SDL 中评估风险并优先修复这些风险。

综上所述，PIA 的目的是提供一些你正在开发的软件中的细节信息，如收集、存储和创建的隐私信息来自于哪里、达到了什么程度。当出现重大决策或者建议使用的软件和范围显著变化时，PIA 应该继续审查和更新。

3.5 安全评估 (A1) 成功的关键因素和度量标准

3.5.1 成功的关键因素

设定成功的标准在 SDL 的任何阶段都会使其更有效，并有助于在事后了解什么工作有帮助而什么没有。表 3-1 列出了作者认为成功的标准。然而，每个环境都是不同的，安全团队都需要在自己的环境中理解成功的标准。

表 3-1 关键成功因素

关键成功因素	描 述
1. 计划 SDL 活动的准确度	所有 SDL 活动被准确地识别
2. 产品风险轮廓	管理部门了解开发产品的真实成本
3. 威胁轮廓的准确度	环境中产品成功的缓解措施和对策很到位
4. 有关规定、认证和法规遵从框架的覆盖	所有适用的法律和法规遵从方面都被覆盖
5. 软件所需安全目标的覆盖	“必须有”的安全目标需要满足

成功因素一：计划 SDL 活动的准确度

该安全评估（A1）阶段是 SDL 的第一阶段，因此大多数都是自然而然发现的。它规定未来的 SDL 活动的基调和方向。正是在这个阶段所需要的 SDL 活动决定什么重点应放在每个 SDL 活动（代码审查、威胁建模等）中。虽然人们总是可以修正方向后再确定 SDL 的活动和它们的重要性，但这一阶段成功的关键措施都是为了最初的需求和 SDL 的方向。虽然这是无法衡量的开始，但是一旦 SDL 周期完成后，应该回到最初的规划文件。这将有助于更准确地估计未来的 SDL 活动。

成功因素二：产品风险轮廓

另一个关键的成功因素是产品风险轮廓。基于软件及其对客户（及其在客户环境中使用）的重要性，通过软件处理的数据，以及有关法规及目标市场 / 国家，对一个产品的基本风险状况可以有所准备。风险轮廓应包括产生于顾客的期望和产品使用的风险，监管服从上的风险，以及以迎合不同市场需要的安全性变更的风险。这也将有助于阐明真正的管理成本。

成功因素三：威胁轮廓的准确度

软件开发没有充分考虑它的预定使用或其将运作的环境的情况十分常见。虽然产品可以用于某些特定用途，但客户经常自行添加增强功能，并且通过那些之前没有充分想过的方式使用。另一个例子是向公众公开的 API。在大多数情况下，API 的暴露增加了一段时间的危险（软件发布后）。然而，暴露这些 API 的所需要考虑的威胁轮廓并不总是被考虑或正确地考虑。另一些情况是，软件依赖在定义产品的威胁状况时没有考虑的开源软件（或闭源代码）。

因此，这一阶段关键的成功因素是威胁状况的准确性。该威胁轮廓应该不仅涵盖感知的用例，还应该依赖其他产品或软件研究客户集成与安全风险。

成功因素四：有关规定、认证和法规遵从框架的覆盖

这一阶段成功的一个关键标准是，所有主要法规、遵从框架，以及产品（或库）的鉴定是否被认证。这一成功因素取决于了解产品的目标和客户使用情况。人们可以很容易地认为某些规定将不适用，因为它们的用例不被视为有效。但是客户往往有不同的看法。从一个视点来看，客户用一个云产品来与其他客户进行交互可能不会需要符合 HIPAA。然而，对于另一个顾客，关键的是，该产品如果不符合 HIPAA，至少不会产生可能导致的问题。

遵从框架是另一件值得注意的事。这取决于不同的产品如何使用（在内部或云中），以及客户预期的不同排列。如果客户打算通过 ISO 27001 认证，并在云环境中使用你的产品，他们会期望一个显而易见且可核查的业务和产品安全态势。如果客户使用信用卡支付您的服务，不仅他们而且你的环境可能处于支付卡行业准则规定的监管下。虽然我们注重产品的安全性，但是运营安全也同样重要。

最后，许多情况下，在涵盖法规、遵从框架和认证时，安全和开发团队没有仔细考虑依赖情况。例如，如果产品是需要符合联邦信息处理标准（FIPS）的，使用一个开放源码库怎么会影响合规性？如果产品需要获得 A 级认证，依赖于它的软件会创造还是打破这种认证？这些问题都需要仔细考虑，以防止未来的安全问题。

成功因素五：软件所需安全目标的覆盖

最后，我们应该看有多少个安全目标在这个结束阶段最终被实现了。如果有些目标没有实现，为什么呢？一开始可能有一个安全目标清单，但是它们经常与其他产品功能竞争，产品管理可能使它们落空。一个例子可能是日志。如果关键的安全目标之一是在威胁发生时探

测和应对它们，有一件事可以帮助做到的就是日志。但是，记录（和安全记录）的可行性可能会与其他要求（运营效率、其他产品功能）矛盾。人们可以通过对日志进行加密，安全地运送它们到一个中央存储库来安全地坚持日志记录事件。但是，根据资源和其他竞争性需求，像日志记录这样的功能可能不会最终实现。

结束本阶段前，看是否有任何安全目标的没有被完全满足是一个好方法；如果没有，确定这些目标是必须实现的，还是最好实现的。这方面的知识将有助于未来产品的 SDL 周期。有太多的“最好实现”实际上可能破坏安全团队的公信力。

3.5.2 可交付成果

在每一个 SDL 阶段，我们将概述该阶段可交付成果的关键集。这样做是为了确保所有必需的活动有一个有形的记录结果。我们常常看到创建并保存一个项目管理团队只是口头或非官方的文件。我们认为，正式文件应建立并保存在一个中央存储库，并且有适当的签名和版本控制。

表 3-2 列出了 A1 阶段关键的可交付成果。

表 3-2 A1 阶段可交付成果

可交付成果	目 标
产品风险轮廓	预估产品的真实成本
SDL 项目概述	将 SDL 映射到开发进度
适用的法律和法规	在适用法律下获得利益相关者的正式批准
威胁轮廓	引导 SDL 活动以减轻威胁
认证要求	列出产品和业务认证的要求
第三方软件列表	识别第三方软件的依赖
度量标准模板	建立定期向主管汇报的节奏

- **产品风险轮廓。**产品风险轮廓帮助管理层从不同的角度看产品的真实成本，包括在不同的市场和债务下出售它（如果它是 SaaS/ 云产品，可能会导致这种状况）。
- **SDL 项目概述（针对安全里程碑和映射到开发进度）。**这个阶段的一个重要结果是 SDL 项目概述或计划。SDL 计划应该包括映射到开发计划 / 进度的安全性里程碑，它们需要在每个阶段得到满足。应建立报告以跟踪该项目进展情况。
- **适用的法律和法规。**这个可交付成果是可能适用于产品的法律和法规全面审查。法律部门应当积极参与编写本文档，并清楚地阐明法律 / 法规不适用的内容，以及为什么这些不适用。
- **威胁轮廓。**此可交付成果阐明关于该产品将经营环境和在环境中的潜在威胁的假设。这将有利于在后期集中我们的 SDL 活动，以确保该产品是在该威胁轮廓下由团队尽可能安全地开发的。它也将对事后分析非常有用，以防我们错过了 SDL 阶段的威胁场景。
- **认证要求。**此可交付成果应明确说明所需的产品（例如，FIPS）和由此产生的需求。在 SaaS/ 云软件的情况下，应确定将需要通过各种框架进行认证的软件的操作控制。
- **第三方软件列表。**这份名单用于找出所有将在我们的软件中使用的第三方组件，从而将其纳入我们的威胁状况。它也应该帮助我们完成所需要的认证更改 / 需求清单。

- **度量标准模板。**该可交付成果是我们计划定期向管理层报告的度量标准模板。

3.5.3 度量标准

在 SDL 模型中，我们提出在每一个阶段应该衡量的指标。但是首先我们想指出在我们的职业范畴内我们已经了解到的几件事情。

我们要决定前期用什么来衡量并尽量遵循这些决策。虽然我们明白，我们可能需要随时时间修改指标，但我们应该抵制时不时地检修指标的诱惑。该指标模板应与受众心目中的放在一起。然而，随着项目的进展，行政人员之间有一种倾向，即要求一组不同的指标。我们应该努力去使行政人员了解指标的选择和它们的重要性。通常，应该关注与指标相互矛盾的建议。概括地说，确定一组适合于你的受众的指标，并坚持下去。从长远来看，指标将为你提供整体进展情况，因此无论选择哪一组指标，它都会满足你的需要。

下面是有关这一阶段度量标准的一些建议：

- 软件安全团队循环的时间（单位：周）
- 参加 SDL 的利益相关者的百分比
- SDL 活动映射到开发活动的百分比
- 安全目标满足的百分比

3.6 本章小结

我们已经描述了在 SDL/ SDLC 过程的最开始解决安全和隐私的重要性与最佳做法。到现在为止，应该明确安全和隐私是得到一个安全的软件开发过程基本层面的要求，定义这两个领域的最佳时间即在本章中所描述的最初规划阶段。定义和建立这些要求使团队能够识别关键的里程碑，并能以对计划和进度干扰最小的方式将安全和隐私集成到软件中。识别 SDL/ SDLC 的关键利益相关者和安全角色，评估和规范软件运行在计划的运行环境中所需的最低安全和隐私要求，总体的 SDL 运行计划，商定的安全漏洞识别 / 整治工作的鉴定项目跟踪系统，是本章所描述的安全评估（A1）阶段的关键要素。但应该明确安全和隐私风险评估是一个 SDL 的强制组成部分。这些都是在定义软件功能时的关键要素，该软件在开发过程后期需要更深的审查。

本章最后讨论了关键成功因素及其重要性，这一阶段的可交付成果，以及应该从这个阶段收集的指标。

本章讨论的最佳做法将作为 SDL 模型后续阶段的基础和基线。下一阶段——体系结构（A2）将在第 4 章继续讨论。

参考文献

1. Kissel, R., et al. (2008), U.S. Department of Commerce, NIST Special Publication 800-64 Revision 2: *Security Considerations in the System Development Life Cycle*. Retrieved from <http://csrc.nist.gov/publications/nistpubs/800-64-Rev2/SP800-64-Revision2.pdf>.
2. Microsoft Corporation (2008), *Privacy Guidelines for Developing Software Products and Services, Version 3.1*. Retrieved from www.microsoft.com/en-us/download/details.aspx?id=16048.

3. Kissel, R., et al. (2008), U.S. Department of Commerce, NIST Special Publication 800-64 Revision 2: *Security Considerations in the System Development Life Cycle*. Retrieved from <http://csrc.nist.gov/publications/nistpubs/800-64-Rev2/SP800-64-Revision2.pdf>.
4. Security Innovation, Inc. (2006), *Regulatory Compliance Demystified: An Introduction to Compliance for Developers*. Retrieved from <http://msdn.microsoft.com/en-us/library/aa480484.aspx>.
5. Microsoft Corporation (2012), "Appendix C: SDL Privacy Questionnaire." Retrieved from <http://msdn.microsoft.com/en-us/library/windows/desktop/cc307393.aspx>.

架构 (A2): SDL 活动与最佳实践

在安全开发生命周期的第 2 阶段，安全方面的考虑被带入软件开发生命周期，以确保所有的威胁、要求、功能和集成方面的潜在制约因素均被考虑到（如图 4-1 所示）。在 SDL 的这个阶段，安全性更多的是看商业风险方面，包括软件安全团队的投入和主要利益相关者在 SDLC 的讨论。商业需求在安全性方面被定义为保密性、完整性和可用性，并且需要讨论创造、传输和个人身份信息（Personally Identifiable Information, PII）的隐私控制。SDL 策略和其他安全或者隐私遵从的要求也在 SDL 的这一阶段确定。这保证了安全和隐私讨论扮演了 SDLC 的一部分，而不是与其分开，因此项目人员会对整体开发项目的商业决定和它们风险的影响有扎实的理解。开发的成本分析、商业需求的安全和隐私一致性所需的支持成本也是需求分析中的一部分。如前所述，计划和先前对 SDLC 中安全、隐私和风险管理的认知，通过在 SDL 中的正确使用，可以显著节约成本和时间。

从这个阶段开始也许是 SDL 中最重要、最复杂和最困难的部分。如前所述，威胁建模和体系结构的安全分析通常需要高级软件安全架构师，并要求对 SDL 中的任务具有最丰富的经验和专业知识。幸运的是，当前可用的或正在开发的过程中的工具可以在此阶段使用，这些工具还能够帮助利用和缩放技能集合，这通常是软件安全团队的限制资源。

关键的开发者可能需要额外的安全培训，以了解他们产品当前的威胁和潜在的风险，同时培训安全设计和编码技术，具体来讲就是正在开发的系统以及与其交互的软件，这些风险和威胁都将在 SDL 的这一阶段被确定。这使得开发人员能够更有效地与软件安全架构师和软件安全组的其他人创建更安全的设计，使他们能够在开发过程的早期解决关键问题。

4.1 A2 策略一致性分析

软件的安全策略的目的是确定哪些需要加以保护以及它如何受到保护，包括审查和结合 SDL 外可能会影响开发进程的策略。这些措施可能包括治理软件或开发或应用在组织中的任何地方的策略。在这个阶段，SDL 的策略域外存在的任何策略都将被审阅。企业安全与隐私策略可能会指示设计和开发人员必须有什么样的安全和隐私功能及它们该如何实现。其他策略可能包括那些支配使用第三方和开源软件或组织内部和外部的保障与控制源代码以及其他知识产权。假设软件安全组独立于集中信息安全组，重要的是，这两个群体就涉及开发和发布后的安全支持及从该组织的软件响应所有的策略和指导方针进行合作。同样重要的是与公司的保密功能进行协作，无论是集中组还是外部法律顾问。

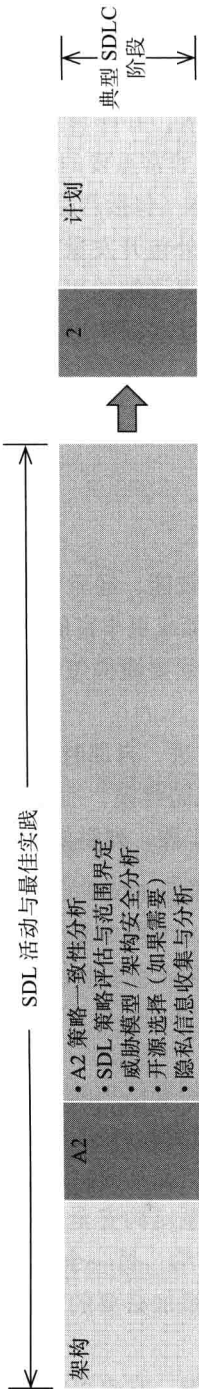


图 4-1 架构 (A2): SDL 活动与最佳实践

4.2 SDL 策略评估和范围界定

SDL 还提供了一个非常宝贵的指南为软件开发人员设置其组织的安全标准，应该提供一个实现路线图而无需中断生产高质量软件应用的核心业务。除非开发组织和管理团队的高层领导支持这种模式，否则 SDL 可能会失败。它必须由签订、出台的政策来驱动，并最好由 CEO 和软件开发管理团队提供支持。一个组织本应该有一个书面的和可重复的 SDL 的策略与方针，以支持 SDLC，其中包括其业务需求，并作为它支持的工程和开发文化的补充。该组织的文化和成熟度在 SDL 策略的制定中是非常重要的，这样可以确保它会同时实现可行性和实用性。管理风格、人员、流程和技术需求（包括产品的整体架构）的复杂性，将有助于确定怎样的粒状或目标来作为重点指导方针。外包开发量，如果有的话，也需要作为这个过程的一部分被评估。一个内部的开发团队需要更详细的程序，而一个外包功能将需要更多的合同对象、服务水平，以及详细的可交付物。这些漏洞以及利用外包资源开发的风险将在本书后面讨论。

4.3 威胁建模 / 架构安全性分析

4.3.1 威胁建模

如前所述，威胁建模需要一套特殊的技能、经验和心态：团队内的人不论是谁做这个一定要像对手一样思考。资深软件安全架构师或更丰富的软件安全冠军之一通常深谙这个方面。参与这个过程的开发者或团队成员不仅必须要懂得怎样开发软件，而且要了解如何解构或拆卸软件及其架构，同时像对手一样思考。

微软首次于 1999 年记载了威胁建模方法，自那时以来，它的方法已经发展成为一个行业标准¹。当然，这不是微软第一次由人来威胁建模，而是第一次将方法形式化或视为一个抽象的工程活动。威胁的风险建模过程有五个步骤，列举如下，并在图 4-2 中显示。它们是：

1. 确定安全目标
2. 调查应用程序
3. 分解它
4. 识别威胁
5. 确定漏洞

下面的这五个步骤将帮助你了解你所需要保护的资产，需要从哪里保护它们，如何保护它们，什么是实现优先级，你将不得不忍受什么风险（如果几个威胁不包括在实施范围内）。

威胁建模的重点不应该是单纯的软件产品本身，而应包括企业的环境和用户。实现优先级可以被限制在完成威胁建模、分析和体系结构安全风险分析之后的软件产品本身。早些在过程中建立安全性不仅可以实现成本的节约，另一个优点是考虑到业务、用户需求和要求，你可以平衡成本效益、提升产品竞争力、添加必要的安全和隐私控制，在此基础上做出安全的决策。

用户上下文不仅影响威胁和漏洞的范围，它也可能强烈影响有关实现的优先级。举例来说，如果你在你的主机环境中存储了客户的信用卡数据，那么数据被窃的威胁，更多来自内部支持人员或雇员，而不是未知社区的外部攻击。从实际威胁建模的角度来看，某些情况下

应该关注以该用户为中心的视图，考虑你的软件产品的授权用户可以访问的任何可能性；是否有同一个未经授权的用户可以升高自己特权的可能性；而已经授权或未经授权的用户是否可以访问管理功能或提供直接访问后端数据库的内容，然后滥用授权。其中一个最坏的情况是，一个未经授权的用户可能危及 Web/ 前端服务器并获取服务器上可用的所有资源升级特权；这将提供利用信任关系获得对重要信息未经授权的访问能力，通过进入 / 事件日志或配置文件。在威胁建模过程中，你必须始终认为自己就像一个攻击者，假设软件产品的所有输入可以是恶意的，而且所有信任边界可在第一层面上突破，也就是软件产品之间的首个互动层和最终用户。²

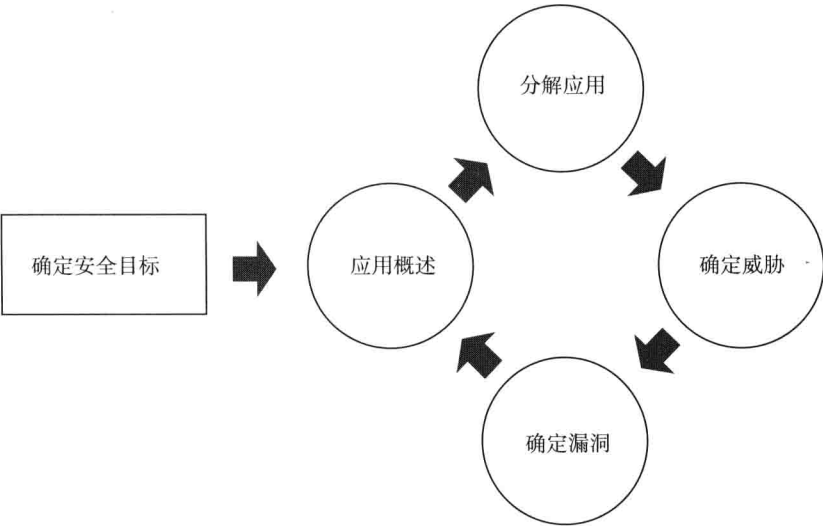


图 4-2 威胁建模的五步骤

威胁建模的目标是通过分解它，理解它如何与外部实体交互，以获得对软件应用的理解。这通过信息收集和归档到一个明确定义的结构完成，从而确保正确的信息收集。从安全的角度看，威胁建模的主要目标是获得关于成功是什么样子的理解，并且为了实现这个目标，你需要安全性成功的基准。这些项目的一个非常有用的清单出现在 2006 年的《MSDN》杂志上，并在这里被转载。

设计原则

- 开放式设计：假设攻击者有来源和规格。
- 故障保护默认值：无法关闭；无单点故障。
- 最小特权：除需要外没有更多特权。
- 机制的经济：保持简单。
- 分离的特权：不要允许基于单一条件的操作。
- 总计调解：每次检查所有事情。
- 最常见的机制：谨防共享资源。
- 心理上的可接受性：他们会使用它吗？

安全性能

- 保密性：数据仅提供给有意访问它的人。

- 完整性：数据和系统资源只以适当的方式由适当的人改变。
- 可用性：在需要的时候和以可接受的方式执行时系统可用。
- 身份验证：确定用户的身份（或者你愿意接受匿名用户）。
- 授权：用户明确允许或拒绝对资源的访问。
- 不可否认性：用户无法执行的操作，之后拒绝执行它。³

参与威胁建模的关键步骤是⁴：

1. 用数据流图打破你的产品架构。
2. 使用 STRIDE 威胁类别，以确定哪些威胁适用于数据流图中的每个元素。
3. 映射适用于使用场景的背景下有关漏洞的所有威胁。
4. 威胁等级评价。为每个威胁和漏洞分配一个风险评级，用于了解它们的影响；这将有助于确定修复它们的优先级。使用 DREAD 或其他方法。
5. 对每个标识的漏洞定义缓解计划 / 对策。
6. 按照前面的步骤确定的优先级修正业务的不能接受的漏洞。

4.3.2 数据流图

威胁建模过程的第一步是确定威胁流的一个可视化表示，通常情况下这以图的形式在白板会话期间绘制。重要的是要对这个过程提供一种结构。提供结构有助于避免错误。如果没有一个良好的图，你可能不会有一个良好的威胁模型。重要的是要了解，第一，这项工作是关于数据流的，而不是代码流。这是团队开发人员常常犯的错误，因为他们的生活、工作与代码开发息息相关，但是通常不注重他们正在开发的代码的数据安全。在这个阶段的威胁建模过程中所产生的图被称为数据流图或 DFD，这应该是毫不奇怪的。DFD 的重点是关注在软件解决方案中数据如何移动，在移动时数据发生了什么，让我们更好地了解软件是如何工作的和它的底层架构，方式是提供软件如何处理数据的可视化表示。该可视化表示是分层的结构，所以它可以让你将软件架构分解成子系统，以及较低级别的子系统。在较高的水平，这可以让你清楚被建模的应用范围；在较低的水平它可以让你专注于处理特定数据时涉及的具体流程。

在开始建立你的 DFD 前，了解你所要使用的元素图像始终是一个好主意。DFD 中通常使用的基本元素和符号如图 4-3 所示。可以通过连接这些不同的元素作为数据流并应用的元素（如适用）之间的边界建立 DFD。

这里使用 DFD 的第 1 个例子是一个 Web 应用程序的威胁建模数据流图，见图 4-4。这个数据流图表示由客户和远程员工从公司网站访问企业营销数据的过程。第一个也是最明显的安全控制区别在公司员工文件访问和客户文件访问之间。员工数据可能包含授权公司仅用于公司员工的 IP 信息，并根据他们的职责，非常敏感的竞争力营销和定价数据只对“需要知道”的员工授权。

图 4-4 的 DFD 图仅供参考，并不代表开发一个应用程序的最佳途径。更仔细检查流程图，我们注意到以下几点：

- 没有员工和客户分割数据。
- 在他们访问网站上的数据之前似乎没有对远程员工（如 VPN）进行双因素身份验证。
- 应该是 Web 应用程序一部分的分层结构尚未开发完全（或至少不属于这个 DFD）。这可

能会简化图，但也可能隐藏着一些用例和流。







元素类型	种类描述	元素符号
外部元素	软件应用外部不受你控制的元素，但是可能会调用和（通过一个入口点建模的）软件有交互	
过程	这个代表软件内部管理数据的任务。这个任务可能会基于数据处理或执行一个任务	
多过程	这个用来表示数据的子过程集合，通常表示另一个 DFD 涉及有故障的子过程或延伸到另外附加的 DFD 中	
数据存储	这个表示存储但没有修改数据的地方	
数据流	这个表示软件中数据的移动过程，箭头表示移动的方向	
信任边界	一个信任边界出现在一个组件不信任边界另一侧的组件时。信任边界通常存在于不同隐私级别的元素之间，当然也可以存在于在同样的隐私级别之间运行的不同组件之间	

图 4-3 DFD 元素类型

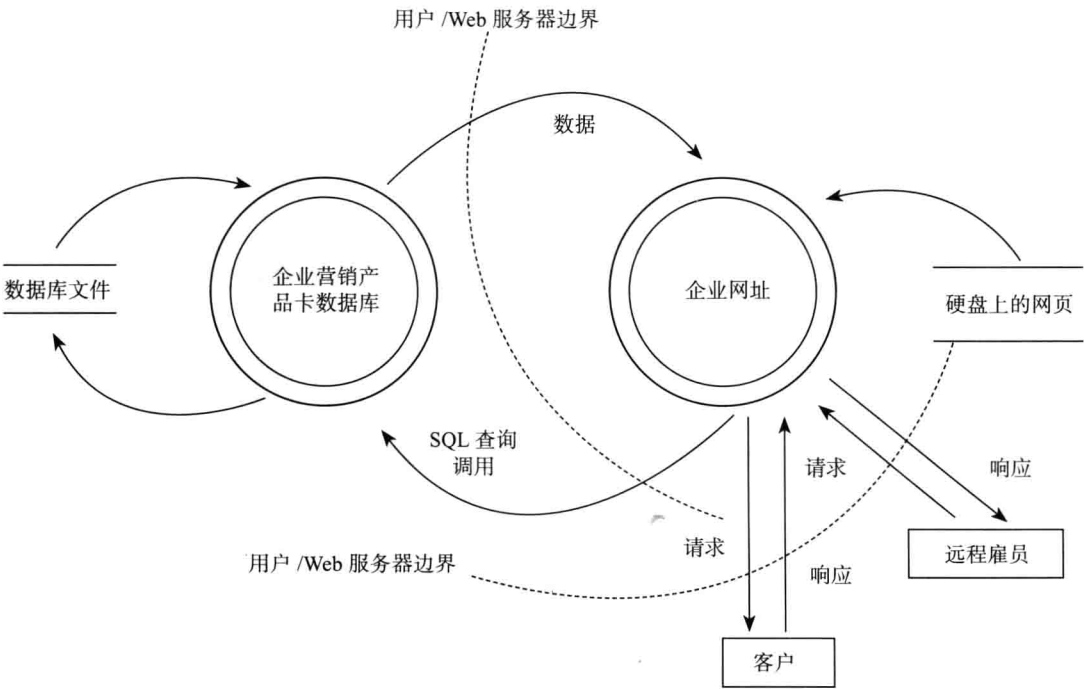


图 4-4 应用程序威胁建模的数据流图示例

图 4-5 中的 DFD 是一个 *aaS 向客户提供基础服务的例子。不同于传统的 Web 应用，这个 DFD 显示了客户怎样通过云服务提供商接入服务的例子。这种情况下，最显著的安全性控制是通过云操作保护客户数据。客户可以通过多种方式，如通过 API 调用、Web 应用程序或

自定义应用程序开发接入服务。

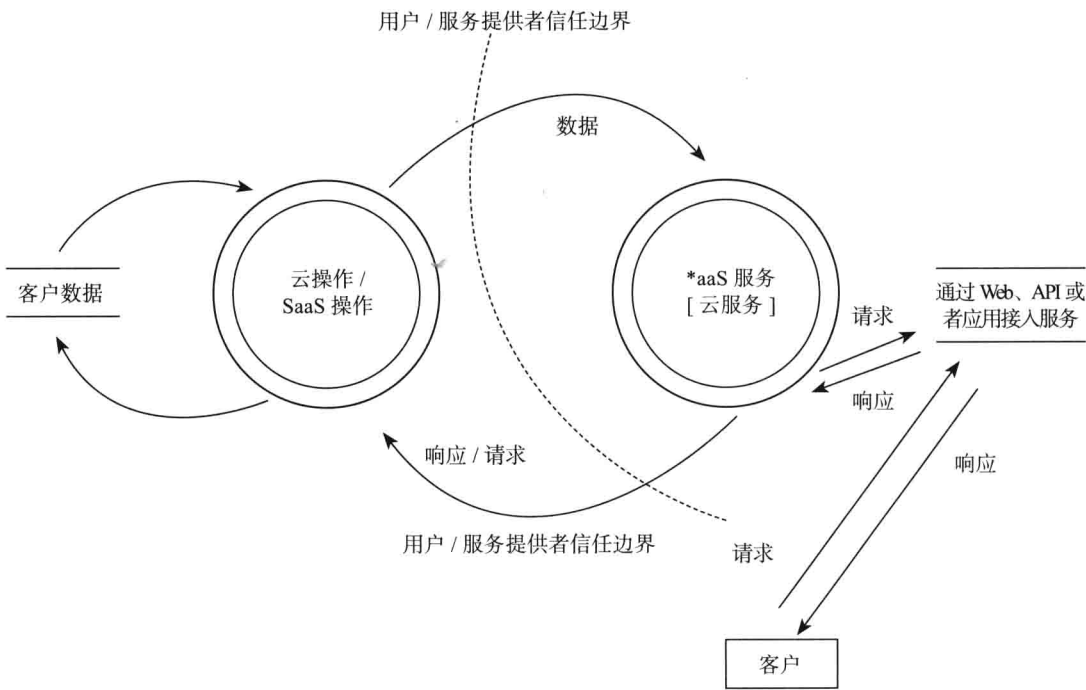


图 4-5 基于云的应用程序数据流图示例

更仔细检查流程图，我们注意到以下几点：

- 应用接入通过 API 或 Web 没有区别。
- 云端操作是更详细的云端操作架构的高层次抽象。
- DFD 不告诉我们不同客户之间的分割。
- 它也没有说明数据多么安全，也就是说，它是否加密？Web 服务器是否只与在集群中的数据库服务器 Web 服务器通信或通信方式是什么？

令 DFD 正确是获得正确的威胁模型的关键。在你的 DFD 上花足够的时间，确保你的系统的所有部分都被表示出来。每个元素（流程、数据存储、数据流和相互交互者）都有一组可能的威胁，就像你可以在图 4-6 中看到的。这个图表，以及你的 DFD，提供了调查你的系统可能会失败的框架。⁵

	数据流	数据存储	过程	外部
欺骗			✓	✓
篡改	✓		✓	
否认		✓ *	✓	✓
信息泄露	✓	✓	✓	
拒绝服务	✓	✓	✓	
提升权限			✓	

图 4-6 威胁影响因素（* 数据存储为日志，有关于不可否认的问题的关注，以及为删除日志而对数据存储的攻击。一组应当使用使评估更加完整的问题，使得这些威胁更加具体和可访问。³⁵）

DFD 过程不仅需要你像一个攻击者一样思考, 而且可能需要像多个攻击者一样思考, 特别是如果你的软件产品是要在云上或 SaaS 环境中运行的话更是如此。一旦 DFD 完成, 你应该对软件如何处理数据有一个准确的概述, 包括它如何移动, 在应用程序和其他与它相关的程序中会发生什么。高水平的 DFD 会阐明应用的范围, 较低水平的 DFD 阐明当处理特定的数据时所涉及的过程。

4.3.3 架构威胁分析和威胁评级

4.3.3.1 威胁判定

判定威胁的第一步是采用一种方法, 通过它可以对威胁进行分类。这提供了在软件应用程序中用一种结构化和可重复的方式系统地识别威胁类别的能力。STRIDE 是威胁分类的一种方法, 该方法多年前由微软普及, 本章将其用作威胁判定工具的一个例子, 但它肯定不是唯一可以使用的方法。STRIDE 中每个字母的缩写有助于归类攻击者的目标:

- 欺骗 (Spoofing, S)
- 篡改 (Tampering, T)
- 否认 (Repudiation, R)
- 信息泄露 (Information disclosure, I)
- 拒绝服务 (Denial of service, D)
- 提升权限 (Elevation of privilege, E)⁶

STRIDE 的第一步是将系统分解成相关的组件, 然后分析每个组件对威胁的敏感性, 最后, 减轻威胁。这个过程重复进行, 直到你接受任何余下的威胁。之后该系统被认为是安全的, 因为你现在已经将你的软件应用程序和系统拆分成单个组件, 并减轻了对每个组件的威胁。当然, 这种方法有其缺陷, 在于所述软件和系统的各个组成部分只是一个较大系统的一部分, 你只能和最薄弱的环节一样安全。软件产品和系统的各个组成部分独立时可能并不易于面对威胁, 但当它是一个较大系统的一部分时则可能会受到威胁。对于那些没有被设计成可以在互联网上、云中、SaaS 环境中使用的软件产品来说尤其如此。

一般威胁和安全控制可影响以下每个 STRIDE 类别。

- 欺骗: 一种威胁行为, 用来非法访问和使用另一个用户的凭证, 如用户名和密码——身份验证
- 篡改: 一种威胁行为, 旨在恶意更改 / 修改持久性数据, 如数据库中的持久性数据, 以及在一个开放的网络中 (如 Internet) 两台计算机之间传输过程中数据的改变——完整性
- 否认: 一种威胁行为, 旨在在一个缺乏追踪被禁止操作能力的系统中进行违规操作——不可否认性
- 信息泄露: 一种威胁行为, 用来读取一个没有授权访问的文件, 或读取传输中的数据——保密性
- 拒绝服务: 威胁的目的是拒绝有效用户的访问, 比如使 Web 服务器暂时不可用或无法使用——可用性
- 提升权限: 威胁的目的是获得访问资源的特权, 获得未经授权的访问信息或破坏系统——授权⁷

4.3.3.2 威胁分析

在完成了DFD之后,你应该找出设计和实现方法进行输入验证、身份验证、授权、配置管理,以及对于其他领域中最容易受到攻击的应用的漏洞,建立所谓的安全配置文件。

下面给出一个实际例子,一般在分析软件应用程序的设计和实施的各个方面会问到以下问题。⁸我们把这些分成这几个大类:输入验证,身份验证,授权,配置管理,敏感数据,会话管理,加密,异常管理,参数操作,以及审计和日志记录。

输入验证

这背后的理由是,高层用户的输入应被视为不可信的,软件在使用前应进行验证。下面是关于输入验证的相关问题:

1. 是否所有的输入数据都要进行验证?
2. 攻击者能否注入命令或恶意数据到应用程序中?
3. 是否需要数据验证,因为它通过不同的信任边界(由接收方入口点)?
4. 在数据库中的数据是否可信?
5. 你想不想将用户输入加入白名单或黑名单?

身份验证

所有的用户交互(和软件/API),与整个系统的交互应被认为验证通过身份验证。没有服务和功能可以无需验证,除非用户/系统/API/组件是合法的。它是否可以使用功能使我们进入授权环节。通常情况下要问的身份验证问题如下:

1. 如果他们通过网络传递是否有担保凭证?
2. 是否使用严格账户策略?
3. 是否强制执行强密码?
4. 是否使用证书?是否在使用任何通配符证书?
5. 是否有密码验证程序(使用单向散列)用于用户的密码?
6. 如何实现系统组件的相互验证(例如如何完成对数据库的服务验证)?
7. 在服务/应用程序的启动过程中,系统组件如何验证对方?
8. 是否使用密钥而不是密码进行身份验证?

授权

很多时候,我们想限制用户/系统/API/组件访问一个软件系统中的某些功能。授权使我们能够做到这一点,即对某些代理防止某些操作。通常涉及授权的问题如下:

1. 用于应用程序入口点的是什么看门人?
2. 在数据库中授权如何强制执行?
3. 是否在使用一个防御纵深策略?
4. 你是否无法安全地访问,只允许在凭证确认成功时访问?

配置管理

配置管理使我们能够强化软件、系统、服务和设备,并锁定下来,从而降低环境风险。配置管理的组成部分包括强化标准和准则,审查应用程序关于服务的依赖性,检查用户和管理员界面,安全变更管理等。问题大致如下:

1. 应用程序支持什么管理界面?
2. 它们是如何确保安全的?

3. 远程管理是否安全?
4. 使用什么样的配置存储? 它们如何保证安全?
5. 是否对于软件栈 (操作系统、数据库、应用程序) 开发了强化的标准?
6. 软件系统是否提供了一种方式从已批准的安全配置更改检测差异?
7. 所有的团队 (IT、QA、工程、运作) 是否对于不同的组件只能使用批准 (金主) 的软件图像, 如操作系统、数据库、Web 和应用服务器?
8. 已批准图像是否用于跨越从开发到部署的整个生命周期?

敏感数据

这方面涉及关于应用和系统处理的数据类型的认识。在许多情况下, 我们发现开发和运营团队都没有意识到或对其应用程序要处理的数据类型, 以及对数据元素的保护是否足够没有足够的认识 (无论是设计或失误)。

1. 应用程序处理什么敏感数据?
2. 什么法规 / 合规性要求适用于数据 / 数据元素?
3. 它是如何保证在网络上和持久性存储中的安全? 这足够符合法律 / 法规要求吗?
4. 使用什么类型的加密? 加密密钥如何确保安全?
5. 敏感数据元素是否出现在日志、源代码或配置文件 (例如, XML) 中?

会话管理

安全地建立和保持注入会话的完整性是当今应用 (尤其是 Web 应用程序) 的关键组件之一。一旦用户通过身份验证, 就会建立会话。这可能会导致在多个场景中的会话被滥用。会话管理的重点是防止这类侵权行为。这方面的典型的问题如下:

1. 会话 cookie 是如何产生的?
2. 它们如何确保安全以防止会话劫持?
3. 如何保证持久性会话状态的安全?
4. 在哪里存储会话信息? 在服务器还是客户端?
5. 当会话跨越网络时, 如何确保会话状态安全?
6. 应用程序如何通过会话存储验证?
7. 证书是由网络传递的吗? 它们由应用程序维护吗? 如果是这样它们如何确保安全?
8. 用户 / 组件的多个会话如何处理?

加密

每个人都使用加密技术。加密倾向于给大多数开发者和用户提供一种安全感。然而, 正确使用加密技术的情况在我们的经历中并不常见。使用加密技术来解决错误的问题往往会使人更加沮丧, 甚至泄露隐私。当使用加密技术处理时, 最好坚持使用经过良好测试的、可公开获得的算法和函数库。关于密码学的问题包括以下内容。

1. 什么是加密技术要解决的问题 (机密性、完整性或两者皆是)?
2. 使用何种算法和加密技术?
3. 是否有任何专属的或内部算法被使用?
4. 加密密钥可以使用多久, 它们是如何确保安全的?
5. 应用程序是否把它自己的加密付诸行动?
6. 密钥被回收的频率是多少? 证书对它进行合法性检查吗? 证书核查撤销列表吗?

参数操作

应用程序经常传递参数与其他方进行沟通。参数范围（不是那么重要）包含迭代器、变量名、值和会话令牌。中间人攻击（MITM）和蓄意篡改参数使得我们检测参数是否确实被安全接收且可以像设计的那样使用的设备机制变得非常重要。当务之急是在接收端（例如输入验证）不要盲目信任参数。

1. 应用程序是否检测被篡改的参数？
2. 应用程序是否仅依赖客户端的验证，还是同时也有服务器端的验证呢？
3. 它是否验证表单字段、视图状态、cookie 数据和 HTTP 包头中的所有参数？
4. 参数是否直接用于数据库查询？
5. 参数是否直接映射给浏览器？

异常管理

优雅地处理错误条件和异常是软件应用的关键。通常情况下，开发者会错过这样的条件或不正确地处理它们。不当的错误处理 / 异常管理的副作用包括拒绝服务或信息泄露。用来探讨这方面的示例如下：

1. 应用程序如何处理错误条件？
2. 是否有一个默认的捕捉异常？
3. 异常是否一直都允许传播回客户端？
4. 一般错误消息是否不包含可利用的信息？
5. 异常日志中是否记录敏感信息？
6. 是编程语言的内置功能用于此目的，还是依赖开发商内部模块？

审计和日志记录

审计和日志在多种原因下都是非常关键的。安全性是其中之一，万一发生法律问题审计追踪很有用。虽然越来越多地注意安全问题，但是操作 / 调试往往是审计 / 日志的驱动程序。以下是一些示例问题，通过这些可以了解审计和日志记录。

1. 请问应用程序审计活动在所有服务器上的所有层次都存在吗？
2. 如何确保日志文件的安全？
3. 应用程序是否记录任何敏感信息（如证书、数据元素、会话令牌）？
4. 日志文件是否安全传输（如 TCP/TLS）？
5. 是否有明确的日志文件保留期限规定？它是否与监管及法律规定一致？
6. 日志多久旋转一次？
7. 对某种类型的事件有触发水平的定义吗？

既然你有了威胁的可视化表示，并回答了上述问题，下一步就是确定可能会影响你的软件应用程序的威胁。这也是你汇集软件安全团队和开发团队的要素，通过白板会议对那些已在威胁建模被确认的漏洞集思广益成本效益和实用的解决方案。攻击者的目标是解决 STRIDE 评估过程中的威胁和问题。这是从一个稍微高一些的架构和多功能的角度来给头脑风暴团队进行包装。使用任何可用的威胁分类列表，并把它应用到任何早期发现的漏洞中去，也是常见的做法。

使用攻击树和攻击模式是一个传统的威胁评估方法，它可以帮助你发现更多潜在的威胁。尽管攻击模式通常表示为攻击，但是它与攻击树结合之后可以用于更深层地分析高亮区域，

这个区域你可能在最初分析的时候错过了，或是它通过使用分类列表发现的已知威胁。由于攻击树是一个层次、结构和流程图的风格，因此它们给了攻击一个很好的可视化表现，并且帮助聚焦潜在的额外方法来避免或减轻此类攻击。它们在创建测试计划和安全成本的评估时也是非常有用的。由于攻击模式、攻击者的技术和 STRIDE 主要侧重于攻击者的目标，所以利用它们与攻击树的结合，有助于给这个过程带来全面的方法，尤其是在面对面的头脑风暴会议中使用它们。

在想要转移到威胁建模和架构风险评估过程的下一阶段并开始给风险评估赋值时，确保你已经解决了关于容易利用性、可能性、影响的风险。在转移到下一个阶段前你必须掌握的只是已经在图 4-7 中给出的可视化的表示。如果没有处理一个领域的风险所需的信息，你需要回去重新完成这个过程，并弥补你的知识和理解的差距。

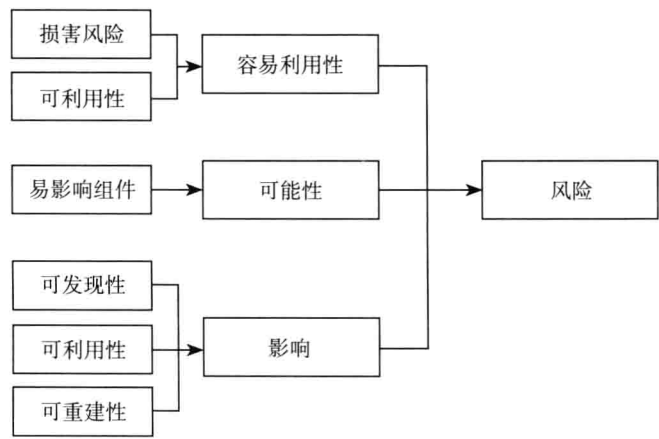


图 4-7 风险评估过程

4.3.3.3 威胁的评级

在威胁建模和架构的安全性分析的最后阶段，从风险的角度来对威胁排名。因为减轻所有识别的威胁可能不是在经济上可行的，所以把它们按风险从最高到最低排名。一些威胁也可以忽略不计，因为除之当它们被利用时只可能导致非常有限的损害之外，它们出现的概率很低。一个由风险确定的威胁优先级列表将极大地帮助确定威胁的优先级别以及缓解的重要性。在较高的层次，这些风险通常会被列为高、中、低三段。在工业中使用的一种典型的风险概率公式表示为：特定漏洞的风险和后果等于威胁的发生概率乘以潜在损害，即：

$$\text{风险} = \text{概率} \times \text{潜在损害}$$

通常在风险概率计算中有 10 级测定，数字 1 代表的是最不可能出现的威胁，数字 10 代表的是最有可能发生的威胁或组件。同样的 1 到 10 的排名系统，用于分配造成破坏的概率，用 1 表示造成破坏的概率最小，10 表示最大。

举一个例子来说明涉及的机制，一个威胁出现的概率为中等，威胁的概率风险分数是 5，潜在损害等于 10，那么它的风险等于一个具有 10 的概率风险分数和潜在损害为 5 的风险。数学表示如下：

如果概率 = 5, 潜在损害 = 10, 那么风险 = $5 \times 10 = 50\%$

如果概率 = 10, 潜在损害 = 5, 那么风险 = $10 \times 5 = 50\%$

正如你可以从这个例子看到的, 100 可以分为三个范围来表示高、中、低风险等级。显然, 你修复漏洞的优先级将从风险优先级最高的漏洞开始, 因为这可能意味着立即解决它是必需的。接下来你将会解决中等风险的漏洞, 那些漏洞需要在此后不久后完成但是优先级稍低。如前所述, 优先级最低的风险取决于工作量、曝光度、财务或法律风险的等级, 当然也与风险相关。

4.3.3.4 DREAD

虽然可以在软件开发过程中评估漏洞时使用许多不同的风险模型, 但是微软使用的 DREAD 模型是最流行的。缩写 DREAD 代表潜在损害 (Damage potential)、可重复性 (Reproducibility)、可利用性 (Exploitability)、受影响的用户 (Affected user) 和可发现性 (Discoverability)。给这些元素产生 1~10 的数字用于建立风险评级; 数字越高, 风险越严重。这些数字用于一个量化、比较和给大量风险排序的分类方案中, 这些风险由每个评估威胁和计算数字形式的整体风险确定, 这样威胁就可以排序并且可以和软件应用程序中找到的其他风险一起排序。

如图所示, DREAD 算法用来计算风险值, 它是 DREAD 类别的平均值:

$$\text{Risk_DREAD} = (\text{DAMAGE} + \text{REPRODUCIBILITY} + \text{EXPLOITABILITY} + \text{AFFECTED USERS} + \text{DISCOVERABILITY})/5^{[9]}$$

下面一些例子展示了如何给一个给定的威胁风险评级, 方法是通过提问来量化 DREAD 类别: ^[10]

潜在损害

- 如果威胁攻击发生, 会造成多大的损害?

■ 0 = 无。

■ 5 = 个人用户数据被泄露或受影响。

■ 10 = 整个系统或全部数据被销毁。

可重复性

- 重现威胁漏洞有多容易?

■ 0 = 非常困难或不可能, 即使该应用程序的管理员。

■ 5 = 需要一个或两个步骤; 可能需要一个授权用户。

■ 10 = 只需要一个 Web 浏览器和地址栏就足够了, 不需要身份验证。

可利用性

- 利用此威胁需要些什么?

■ 0 = 高级编程和网络知识, 具有定制或高级的攻击工具。

■ 5 = 互联网上存在恶意软件, 或一个漏洞利用攻击工具很容易进行。

■ 10 = 只需一个 Web 浏览器。

受影响的用户

- 有多少用户将会受到影响?

- 0 = 无。
- 5 = 一些用户，但不是全部。
- 10 = 所有用户。

可发现性

- 发现这种威胁有多么容易？

- 0 = 非常困难或不可能；需要源代码或管理员权限。
- 5 = 通过猜测或监控网络痕迹可以识别。
- 9 = 像这种错误的细节已在公共领域出现，使用搜索引擎可以很容易发现。
- 10 = 信息在 Web 浏览器地址栏或表单中是可见的。

下一步是根据你的回答给每一个 DREAD 类别划分威胁等级：低（值=1）、中（值=2）和高（值=3），如下所示。^[11]

潜在损害

低（值=1）：泄露琐碎的信息。

中（值=2）：泄露敏感信息。

高（值=3）：攻击者可以颠覆安全体系；得到充分信任的授权；以管理员身份运行；上传内容。

可重复性

低（值=1）：这种攻击是非常难以再现的，即使有安全漏洞的知识。

中（值=2）：该攻击可以重复，但只有一个计时窗口和一个特定的竞争情况。

高（值=3）：该攻击每次都重复，并不需要一个计时窗口。

可利用性

低（值=1）：攻击需要一个非常熟练技能的人，并深入了解每一次可以利用漏洞攻击的知识。

中（值=2）：一个熟练的程序员就可以攻击，然后重复上述步骤。

高（值=3）：一个初学者在很短的时间内就可以攻击。

受影响的用户

低（值=1）：很小比例的用户，模糊的功能；影响匿名用户。

中（值=2）：一些用户，非默认配置。

高（值=3）：所有用户，默认配置，重点客户。

可发现性

低（值=1）：这个 bug 是模糊的，而用户是不可能计算出潜在损害的。

中（值=2）：该漏洞位于产品中一个很少使用的部分，只有少数用户会碰到它。这需要一些思考，看看是否有人恶意使用。

高（值=3）：发布信息解释了该攻击。该漏洞位于最常用的功能中，是非常明显的。

这些数字可以放入类似于图 4-8 所示的一个矩阵中。之后对于一个给定的威胁值要进行统计和求和，其结果将落入 5 ~ 15 范围内。整体评级为 12 ~ 15 的威胁通常具有高风险，那些评级为 8 ~ 11 的威胁具有中等风险，那些评级为 5 ~ 7 的威胁具有低风险。

DREAD 类别	低风险（1）	中等风险（2）	高风险（3）	小计风险分数
潜在损害（D）				
可重复性（R）				
可利用性（E）				
受影响的用户（A）				
可发现性（D）				总计风险分数

图 4-8 DREAD 威胁等级表

4.3.3.5 Web 应用程序安全框架

Web 应用程序安全框架也称为应用程序安全框架（Application Security Frame, ASF），它使用分类来组织常见的安全漏洞，尤其专注于 Web 软件应用程序。当你在审查应用程序设计以创建一个威胁模型时，如果使用这些分类，你可以系统地揭示应用程序架构的威胁和漏洞。总共有 9 种框架分类；在这个过程中使用的样例问题如下所示。

Web 应用程序安全框架分类和评估问题^[12]

• 输入和数据验证

你怎么知道应用程序接收的输入是有效和安全的？输入验证是指额外的处理之前，应用程序如何过滤、清洁或拒绝输入。考虑通过入口点限制输入，并通过出口点编码输出。你是否信任你的数据来源，如数据库和文件共享数据？

• 身份验证

你是谁？身份验证是一个实体证明另一个实体的身份的过程，通常通过证书来实现，例如用户名和密码。

• 授权

你能做些什么？授权是应用程序的资源和操作是如何提供访问控制的。

• 配置管理

应用程序以什么身份运行？它连接到哪个数据库？应用程序是如何管理的？这些设置如何保证安全？配置管理是指应用程序如何处理这些业务问题。

• 敏感数据

应用程序如何处理敏感数据？敏感数据是指应用程序如何处理必须加以保护的数据，不论是在内存中，在网络上，还是在持久性存储中。

• 会话管理

应用程序如何处理和保护用户会话？会话是用户和 Web 应用程序之间的一系列关联交互。

• 加密

如何保持机密（保密性）？如何防止篡改数据或数据库的（完整性）？对于必须强加密的随机值如何提供种子？加密是指应用程序如何强制执行机密性和完整性。

• 异常管理

在应用程序中，当一个方法调用失败时，应用程序是怎么做的呢？你透露了多少？你向最终用户返回友好的错误信息了吗？你把有价值的异常信息传递回调用者了吗？应用程序优雅地出现故障吗？

• 审计和日志

谁在什么时候做了什么？审计和日志是指应用程序如何记录与安全相关的事件。

4.3.3.6 通用风险模型

微软的威胁建模过程（如 STRIDE 和 DREAD）可能不适合你的应用程序，你可能希望使用其他的威胁风险模型或者修改微软威胁建模过程以供自己使用，对于自己的组织使用最合适的威胁建模方法。采用定性值，如高、中、低，还可以避免排序变得过于主观，就像 DREAD 所使用的数字系统。

这些例子通过分给可能性和影响因素配定性值（如高、中、低），有助于整体风险值的计算。在这里采用定性值，而不是像 DREAD 模型中的数字，也有助于避免排名变得过于主观。

一个更主观的模型例子是通用风险模型，其中考虑到概念（例如，攻击的概率）和影响（例如，潜在损害），并用数学表示如下：^[13]

$$\text{风险} = \text{概率} \times \text{影响}$$

可能性或概率是由威胁是否容易利用来确定的，这主要取决于威胁的类型和系统的特性，并通过考虑实现一种威胁的可能性，这是由一个适当对策的存在性来定义的。下面是用于确定威胁易利用性的一组考虑因素：

1. 攻击者能否远程利用此威胁？
2. 攻击者是否需要进行身份验证？
3. 漏洞可以自动地利用吗？

影响主要取决于潜在损害和影响范围，如包括受威胁组件的数量。下面举例来确定潜在损害：

1. 攻击者能否完全接管和操纵系统？
2. 攻击者能否获得管理权限访问系统？
3. 攻击者是否可以导致系统崩溃？
4. 攻击者是否可以取得访问敏感信息的权限，如机密、PII？

确定受威胁影响的组件的数量的示例包括：

1. 有多少数据源和系统可能会受到影响？
2. 威胁代理可以多么“深”入基础设施？

4.3.3.7 Trike

Trike 是一种不同于 STRIDE 和 DREAD 的威胁建模方法。Trike 是一个统一的概念性框架，它从一个风险管理的视角，通过以可靠、可重复的方式产生威胁模型，用以完成安全审计。^[14] Trike 使用威胁建模框架（类似于微软的威胁建模方法）。然而，Trike 的不同之处在于它使用具有独特实施、威胁和风险模型且基于风险的方法，而不是使用 STRIDE/DREAD 汇总威胁模型（攻击、威胁和缺陷）。^[15]

Trike 区别于其他威胁模型方法的地方是从该系统的防御性角度和方法论形式主义的程度，使其系统尽可能高度自动化。^[16] Trike 工具的最新版本可以从 Source Forge 网站下载：<http://sourceforge.net/projects/trike/files/trike>。

一个安全审计团队可以用 Trike 来描述系统的安全性特点：从高层次的架构到其低层次的实现细节。Trike 的目标是将威胁建模的重复部分自动化。Trike 自动生成基于所述系统描

述的威胁（和一些攻击），但是这需要用户向 Trike 描述系统，并检查这些威胁和攻击是否适用。^[17] Trike 的一个关键要素是授权、参与、并与关键的利益相关者沟通完整进度和任务状态透明性，便于他们知道风险水平，并可以评估在整个软件开发过程中可以接受的风险。

4.3.3.8 PASTA

2011 年，Marco Morana 和 Tony Uceda Velez 开发出一种新的应用程序威胁建模方法。攻击仿真与威胁分析过程（Process for Attack Simulation and Threat Analysis, PASTA）是一个包括 7 步骤的过程，它适用于大多数应用程序开发方法，并且是平台无关的。它不仅对比业务目标与技术要求，同时也考虑到了合规性要求、业务影响分析、威胁管理的动态方法、枚举和评分。这个过程从业务目标、安全性和法规遵从要求、业务影响分析的明确的定义开始。类似于微软的过程，应用程序被分解成组件，使用用例图和 DFD 说明威胁模型有哪些威胁和漏洞分析可以执行。下一步涉及对于分析中的进一步引用威胁树、滥用案例、评分系统，以及枚举。此后，从攻击者的角度，威胁模型通过使用威胁树攻击建模和攻击面分析查看。在最后一步中，风险和业务影响将被定性化和定量化。这一过程结合了最好的威胁建模方法，用攻击树作为查看威胁的以攻击者为中心的方法，以及把风险和影响分析相结合，打造出以资产为中心的缓解策略。威胁树与现有漏洞的映射，有利于扩展威胁管理工作。除了技术方面的问题之外，风险和业务影响分析，使得威胁建模不仅仅是一个软件开发工作（涉及关键决策者在漏洞管理过程中的参与）。它侧重于该过程最后阶段的风险管理措施。这保证了它并不局限于特定的风险估计公式。^[18]

7 步 PASTA 威胁建模方法如下。^[19]

1. 定义目标。

- 确定业务目标。
- 确定安全性和法规遵从要求。
- 业务影响分析。

2. 定义技术范围。

- 获取技术环境的边界。
- 获取基础设施、应用程序和软件依赖性。

3. 应用分解。

- 确定用例，定义应用程序入口点和信任级别。
- 确定参与者、资产、服务、角色和数据源。
- 数据流图表和信任边界。

4. 威胁分析。

- 概率攻击情况分析。
- 对安全事件的回归分析。
- 威胁相关的情报和分析。

5. 漏洞和缺陷分析。

- 现有的漏洞报告和问题跟踪查询。
- 使用威胁树威胁到现有的漏洞映射。
- 通过用例和滥用案例设计缺陷分析。
- 打分（CVSS/ CWSS）和枚举（CWE/ CVE）。

6. 攻击建模。

- 攻击面分析。
- 攻击树的开发和攻击库管理。
- 使用攻击树进行漏洞攻击和漏洞利用分析。

7. 风险及影响分析。

- 验证和量化业务影响。
- 对策识别和剩余风险分析。
- ID 风险缓解策略。

有一种新的威胁建模工具——ThreatModeler，它由 MyAppSecurity 公司开发，支持 PASTA 方法。ThreatModeler 是一种威胁建模产品，它使用思维导图的方法来进行威胁建模。它使公司能够轻松、毫不费力地扩展数以千计的威胁建模活动。ThreatModeler 自动产生威胁，并根据它们不同的风险类别进行分类。它提供了一个集中式威胁管理平台，企业可以在该平台中定义与网络、主机、应用、手机、Web 服务等有关的威胁以及关联的属性，如技术影响、业务影响和威胁代理，以便更好地了解威胁和指定缓解策略优先级。

虽然原来的重点是应对银行恶意软件的威胁，但是 PASTA 有跨应用软件的适用性，将很好地适应大多数 SDLC。看看它在未来几年是如何被广泛接受的将是非常有趣的。

4.3.3.9 CVSS

另一种风险评估方法是美国非常流行的政府通用漏洞评分系统（Common Vulnerability Scoring System, CVSS），企业产品安全事件响应小组（PSIRT）和内部软件安全团队广泛使用 CVSS 来对外部发现的软件漏洞进行分类。受美国国家基础设施顾问委员会（NIAC）的委托，CVSS 支持全球漏洞披露框架。CVSS 目前是由事件响应和安全小组论坛（FIRST）维持的，过去由许多公司（包括 CERT / CC、思科系统、DHS/MITRE、易趣网、Internet Security Systems、微软、Qualys 和 Symantec）共同维护，旨在为最终用户提供一个综合评分（表示漏洞的严重性和风险）。它由指标和公式推导得出。该指标分成三个不同的类别，可以是定量或定性测定的。基础指标含有任何给定漏洞内部的量；这些量不会随着时间的推移或在不同的环境中而变化。时间指标包含了随着其生命周期变化的漏洞的特征。环境指标包含在特定用户的环境中与实现相关的漏洞的特征。评分是根据特定公式结合所有度量值的过程，它基于一系列的指标（基于专家评价指标），其描述如下。^[23]

- 基本得分是由供应商或发起人为了发布而计算出的，并且一旦设置好，就预计不会改变。基本得分也从机密性、完整性和可用性的方面来计算。这是时间和环境指标修正的基础。基础得分与最后的得分关系最大，并且代表了漏洞严重性。
- 时间得分也由供应商和协调员计算和发布，并修改基础分数。它可以介绍缓解因素以降低漏洞评分，并且在特定的时间间隔作为一个漏洞的年限重新评估。时间得分代表在特定时间点上漏洞的紧迫性。
- 环境得分由最终用户组织计算并依据基础得分和时间得分调整综合得分。这种综合得分应被视为最后得分，并代表某一时刻针对特定环境的得分。用户组织应该在自己的环境中使用这个分数来判断响应的优先级。

Common Vulnerability Scoring System 2 Calculator 是一个有用的工具，它可以在美国国家标准与技术研究院（NIST）的国家漏洞数据库网站看到，网址为 <http://nvd.nist.gov/cvss>。

cfm?calculator&version=2。

CVSS 已成为评估计算机系统安全漏洞严重性的行业标准。它确立了关注漏洞的措施（与其他漏洞相比），哪个缓解工作可以优先。写作本书时，CVSS 的版本为 2。

内部软件安全团队通常使用 CVSS 来回应安全研究人员或其他来源提出的软件漏洞。这有助于让漏洞严重性等级归一化、一致和准确。分数也用于与客户交流，承认他们已经购买的产品中有漏洞，该漏洞的严重性，以及你的公司做了什么去减轻该漏洞，包括发布漏洞的补丁。反过来，一名安全研究人员可能会使用 CVSS 评级系统为在其软件产品中发现漏洞的公司提供一个风险评级，从而使供应商对被披露的漏洞有更好的认识，让他们的产品开发团队熟悉需要验证的细节。

应当指出的是，CVSS 不是威胁建模方法，它并不用来查找或减少攻击面或帮助指定的一段代码风险。这是一个风险评分系统，它增加了 STRIDE 和 DREAD 中不存在的复杂性。它用来计算除了环境因素之外产品发布后的风险。

4.3.3.10 OCTAVE

OCTAVE（操作关键威胁、资产和漏洞评测）是一个非常复杂的风险方法，由卡内基梅隆大学软件工程研究所（SEI）和 SEI 计算机应急响应小组（CERT）合作发起。OCTAVE 关注组织风险，而不是技术风险。它包括一套工具、技术以及基于风险的信息安全战略评估与规划方法。有三个 OCTAVE 方法：（1）原始 OCTAVE 法（构成 OCTAVE 基础知识的主体）；（2）OCTAVE-S（针对较小的组织）；（3）OCTAVE-Allegro（信息安全评估和保证的一个简化方法）。所有的方法都有具体的做法、配置文件和目录记录建模结果的特定工作表。OCTAVE 方法建立在 OCTAVE 标准上，它是风险驱动的并以实践为基础的信息安全评估的一个标准方法。OCTAVE 标准确立了 OCTAVE 方法使用的基本原则和建立风险管理属性。^[24]

OCTAVE 是一个有价值的结构化方法，用来记录和测量整体 IT 安全风险，特别是因为它涉及企业的 IT 和业务风险管理，以及记录完整系统周围的风险时变得必要。虽然软件安全专家可参与评估，因为在开发过程中也许应该建立安全的软件或过程，但它对于在 SDL 流程中针对特定风险与漏洞建模、定义和排名是没有价值的。与 CVSS 评分相同，OCTAVE 不包括威胁风险建模，它主要用于枚举风险。它也比其他大多数风险评估和打分方法复杂得多。OCTAVE 的综合版本（不同于小型组织的 OCTAVE-S）定义的“可能性”假设威胁总是会发生，这并不适用于许多大型组织。由于这些原因，它是一种不太可能用于整个软件开发生命周期的方法。

4.3.3.11 AS/NZS ISO 31000:2009

澳大利亚 / 新西兰标准 AS/NZS 4360 于 1999 年首次发布，2004 年修订，是世界上记录和管理风险第一个正式标准，也是管理风险为数不多的正式标准之一。^[25] AS/NZS ISO 31000:2009 是 2009 年 11 月 20 日发布的一个较新标准，用于管理风险并取代 AS/NZS 4360:2004。^[26]

ISO 31000:2009 提供了风险管理的原则和一般准则，可用于任何公共、私人或社会企业、协会、团体或个人，而不是针对任何行业或部门。它可以应用于整个组织的生命周期，并可以应用于广泛的活动，包括战略和决策、操作、流程、功能、项目、产品、服务及资产。它也可以应用到任何类型的风险，不论其性质，不论具有正面或负面后果。尽管 ISO 31000:2009 提供了通用准则，但是它的目的不是要促进整个组织风险管理的一致性。风险管理计划和框架的设计和实施都需要考虑到一个特定组织不同的需要，其特定的目标、背

景、结构、操作、流程、功能、项目、产品、服务或资产，以及具体做法的不同需求。ISO 31000:2009 用来协调现有的和未来标准中的风险管理流程。它提供了一个通用方法，以支持应对特定风险和 / 或行业标准的管理，但并不会取代这些标准。^[27]

ISO 31000:2009 没有定义方法来进行结构化的威胁风险建模，或结构化的方法来指定应用软件的安全风险，并评估业务或系统性风险，而不是技术风险。正如 OCTAVE，这种方法很可能由一个集中的企业风险管理团队来使用。虽然软件安全专家可参与评估，因为用于在开发过程中建立软件安全性的软件或过程也许在其使用范围内，但它是不太可能是用于 SDL 中的主要风险方法，因为它在 SDLC 流程中对于特定风险和漏洞建模、定义和排名是没有价值的。

4.3.4 风险缓解

在您移动到风险缓解期前，在威胁建模过程中需要创建一个包括 STRIDE 或其他类似方法的高风险漏洞的主列表。这将给你一个优先列表，你可以按照这个列表制定缓解计划。

有四种方法可以计划减缓和应对威胁：^[28]

1. 重新设计过程以消除威胁。
2. 作为一般性建议应用标准的缓解方法。
3. 发明一种新的缓解策略（高风险且耗时）
4. 接受低风险、非常费劲才能解决的安全漏洞。

对那些在 STRIDE 或应用程序安全框架（ASF）无对策的漏洞威胁进行分类，并对每个类别中特定的威胁给出对策。OWASP 有两个清单，虽然没有包容性，但可以作为这一活动的重大方针。这两个列表都可以在 https://www.owasp.org/index.php/Application_Threat_Modeling 找到，并且在下面对相应的缓解技术进行了描述。

STRIDE 威胁和缓解技术列表^[29]

伪造身份

- 适当的身份验证
- 保护机密数据
- 不要存储机密

篡改数据

- 适当的授权
- 散列
- MAC
- 数字签名
- 防篡改协议

否认

- 数字签名
- 时间戳
- 审计跟踪

信息披露

- 授权

- 保密增强协议
- 加密
- 保护机密
- 不要存储机密

拒绝服务

- 适当的身份验证
- 适当的授权
- 过滤
- 节流
- 服务质量

特权提升

- 使用最低权限运行

ASF 的威胁和对策列表^[30]

身份验证

1. 在存储和运输中对证书和身份验证令牌进行加密保护。
2. 协议可以抵抗暴力破解、字典和重复攻击。
3. 强制执行强密码策略。
4. 用可信服务器身份验证来代替 SQL 身份验证。
5. 用散列方法 (salted hash) 存储密码。
6. 重设密码时不泄露密码的提示和有效的用户名。

授权

1. 严格的 ACL 用来强制授权访问资源。
2. 基于角色的访问控制用来限制访问特定的操作。
3. 该系统遵循用户和服务账户的最低权限原则。
4. 特权分离在表示层业务层和数据访问层中正确配置。

配置管理

1. 最小特权进程的使用和服务账户没有管理能力。
2. 启用所有管理活动的审计与日志记录功能。
3. 配置文件和管理员界面的访问仅限于管理员。

存储和传输中的数据保护

1. 使用标准的加密算法和正确的密钥长度。
2. 散列的消息认证码 (HMAC) 用来保护数据的完整性。
3. 机密 (如密钥、机密数据) 无论是在运输和存储过程中都被加密保护。
4. 内置安全存储用于保护密钥。
5. 无凭据和敏感数据以明文形式发送。

数据验证 / 参数验证

1. 强制检查数据类型、格式、长度和范围。
2. 从客户端发送的所有数据将被验证。
3. 没有安全决定基于可操作的参数 (例如, URL 参数)。

4. 使用通过白名单验证的输入过滤。

5. 使用输出编码。

错误处理和异常管理

1. 所有异常使用结构化的方式进行处理。

2. 在发生错误和异常的时候权限都将恢复到适当水平。

3. 错误消息被清洗，这样就没有任何敏感信息被泄露给攻击者。

用户和会话管理

1. 没有敏感信息明文存储在 cookie 中。

2. 身份验证 cookie 的内容进行加密。

3. cookies 配置到期。

4. 会话能够抵抗重放攻击。

5. 安全的沟通渠道用来保护身份验证 cookie。

6. 执行关键功能时，强制用户重新进行身份验证。

7. 会话过期时注销。

审计和日志记录

1. 不记录敏感信息（如密码、PII）。

2. 访问控制（例如，ACL）被强制记录在日志文件中以防止未经授权的访问。

3. 完整性控制（例如，签名）被强制记录在日志文件中以提供不可抵赖性。

4. 日志文件为敏感操作和关键事件的记录提供审计线索。

5. 审计和日志记录在各层启用的多台服务器上。

在你已经确定了软件的威胁和相关的缓解战略以及对风险进行了排名后，对于威胁建模 / 架构安全分析过程中发现的每个威胁就有可能给出一个缓解威胁的框架。下列标准是用来给最终名单完成分类的。

- **完全缓解的威胁：**威胁具有相应的对策，并且不再暴露漏洞或造成影响。
- **部分缓解的威胁：**威胁使用一个或一个以上的对策被部分缓解，这表示可以漏洞仅可以被部分利用，并且造成有限的影响。
- **未缓解威胁：**威胁没有对策，并表示可以得到充分利用的漏洞，并造成影响。^[31]

既然你已经将你的威胁归类到在上述三类中的一个，你就可以选择对应的战略方法。可以选择以下五个操作之一：

1. 什么也不做，例如：抱最好的希望。
2. 了解有关的风险：例如，警告有关风险的用户群。
3. 降低风险：例如，通过实施应对措施。
4. 接受风险：例如，评估威胁被利用的影响（业务影响）。
5. 转移风险：例如，通过合同协议和保险。^[32]

该使用上述策略中哪一个取决于如下几个因素。

- 某一威胁被利用造成的影响。
- 发生的可能性。
- 转换的成本（即保险成本）或者避免成本（即重新设计的成本或损失）。^[33]

在这个意义上，风险是一个确定威胁的情况（由于潜在存在的参与者、动机以及具有显

著概率的和业务影响的漏洞)。威胁的风险不是问题,而标识具有显著的业务影响的严重风险才是问题。威胁的概率是分开考虑的,因为它是受参与者的动机和具体细节,以及影响漏洞的外部因素。业务影响也是风险一个重要组成,它被参与者的类型(可以是国家、产业或罪犯)以及漏洞的具体细节所影响。这可以用图 4-9 来表示。

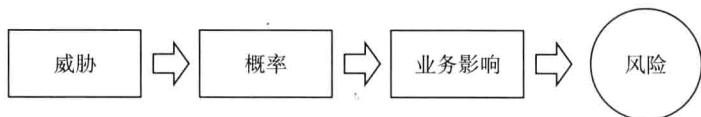


图 4-9 风险的元素

本节已经给出了一些可以用来评估威胁,排名风险,并制定风险缓解计划的选项和标准方法,该过程如图 4-10 所示。

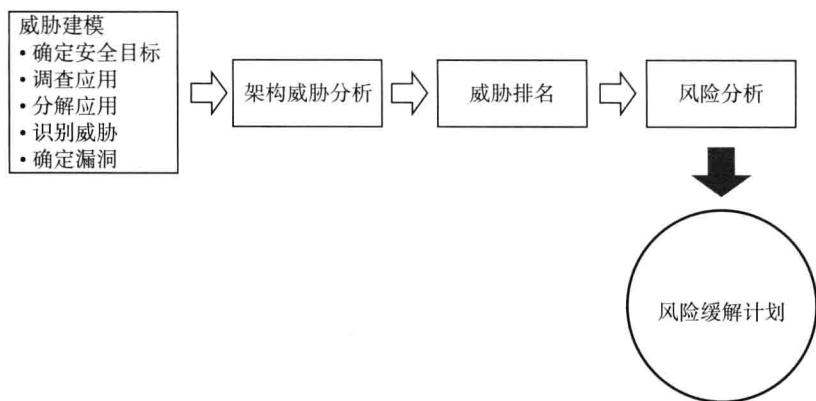


图 4-10 软件安全风险评估的一种整体方法

底线是风险评估是对企业经营风险和结果的取舍,因为它涉及软件的安全风险,系统与它的交互,以及整体业务风险管理策略。安全专业人士了解这一点是很重要的,这就是为什么我们使用两种主流整体业务风险评估方法,以专注于信息安全风险,尤其是 OCTAVE 和 AS/NZS ISO 31000:2009。与软件是生态系统的一部分类似,用于安全软件开发的风险评估方法也会影响与整体业务风险管理方法的交互,这些领域是软件风险分析需要考虑的一部分。

最终,这将是一个商业决定。这就是为什么只有在修复漏洞的成本低于利用此漏洞导致潜在的业务影响时,决定才可能是修复漏洞。这也就是为什么还有另一项决定:接受风险(当缺失一些安全控制(如机密性、完整性和可用性)的风险只涉及一小部分服务而不会丧失一个关键的业务功能时)。

4.4 开源选择

目前软件行业有日益增加的趋势,在过去几年中开源软件和专有软件以最低的成本提供最高价值的服务。两者的融合称为“混源”,这已经成为行业的主导做法。因为开源成为软件开发领域越来越大的部分,所以了解和管理软件资产的授权将是至关重要的,但是这超出了我们的讨论范围,这将会被别的软件开发团队处理。

关于开源软件会增加软件的安全性还是会不利于它有一个正在进行的讨论,但底线是,

你要导入软件到你的软件应用程序或解决方案中，而你的公司没有开发或者安全监督权。这将需要一个广泛的审查——通常称为第三方安全评估，该评估将由你的软件安全架构师、第三方组织，或两者的结合来进行。虽然很容易依靠工具和开源开发过程的粗略审查。但是如果没有适当的训练和经验，很容易得到错误的结果，难以形成一个可操作的修复策略。这就是高级软件安全架构师或第三方组织必须参与这一审查过程的原因。他们有多年的代码安全审计经验，有定期审查高度复杂和先进的软件安全和架构的挑战，知道如何识别和检查设计中的薄弱环节，并能发现可能导致安全的缺陷。如果没有适当的训练和经验，很容易导致错误，并且难以形成必要的可操作的补救策略。从本质上讲，对软件产品中使用的任何开源软件或组件的审查，都需要由经验丰富的软件安全架构师进行工具评估、进一步的威胁建模和风险评估。

4.5 隐私信息收集和分析

考虑到系统是否能够传输、存储、创建隐私信息在 SDLC 早期是很重要的。信息收集、辨识以及规划实施适当的保障措施和安全控制（包括解决隐私信息事件处理和报告需求的流程）都是在这个阶段决定的。SDL 的这个阶段就是对隐私影响评估（PIA）的信息收集和分析的开始。分析阶段决定了 PII 将如何处理，以确保其符合有关隐私的法律、法规和政策规定；软件和开发的整个系统中，收集、维护、以可识别的形式传播隐私信息的风险和影响，或在云或 SaaS 环境中与它交接的风险和影响。检查和评估保护和处理用于减轻潜在的隐私风险的信息的替代过程。

4.6 成功的关键因素和度量标准

4.6.1 成功的关键因素

SDL 第二阶段的成功取决于 SDLC 如何同步识别威胁、要求，约束功能和集成，以及减轻风险。第二阶段的关键成功因素见表 4-1。

表 4-1 关键成功因素

关键成功因素	描 述
1. 确定业务需求和风险	按 CIA 定义的业务需求和风险的映射
2. 有效的威胁建模	识别软件威胁
3. 有效的架构威胁分析	分析软件威胁和威胁出现的概率
4. 有效的风险缓解策略	每一个业务需求的风险接受、容忍和缓解计划
5. DFD 的准确度	威胁建模中使用的数据流程图

成功因素 1：确定业务需求和风险

在这个阶段，关键的利益相关者，包括软件安全团队明确了业务风险和要求。业务需求通过信息安全的 CIA 支柱定义。成功 SDL 周期的当务之急是，尽可能确定所有的要求并且捕获需求。

成功因素 2：有效的威胁建模

有效的威胁建模中是一个复杂和艰巨的任务，威胁建模的整个风险缓解计划取决于这个任务。威胁模型的任何间隙会导致软件和 / 或部署中缺乏有效的安全控制。

成功因素 3：有效的架构威胁分析

架构威胁分析可以识别威胁并按优先顺序排列它们。至关重要的是，所有的威胁向量导致风险的确定和优先级排列。

成功因素 4：有效的风险缓解策略

威胁建模和威胁分析的关键是风险接受度、宽容度和风险缓解计划。至关重要的是，企业的风险接受度和宽容度将进行彻底审查，包括通过法律和财务的方法。

成功因素 5：DFD 的准确度

DFD 是在威胁建模中用于识别感兴趣的各组件 / 元素的方法。DFD 应尽可能详细。任何假设都应该仔细审查。具体来说，信任边界（客户端 / 服务器、私有 / 公共基础设施、分层架构）等应妥善记录和审查。

4.6.2 可交付成果

表 4-2 列出了 SDL 这个阶段的可交付成果。

表 4-2 A2 阶段的可交付成果

可交付成果	目 标
业务需求	软件需求，包括 CIA
威胁建模工件	数据流图、元素、威胁列表
架构威胁分析	基于威胁分析的威胁和风险的优先级
风险缓解计划	计划去缓解、接受或者容忍风险
政策符合性分析	遵守公司政策的分析

业务需求

正式的业务需求是一个工件，它列出软件需求和业务风险，这些需求和风险映射到信息安全的三大支柱：机密性、完整性和可用性。

威胁建模工件

这是 SDL 阶段的一个重要组成部分，有一些工件来自该步骤。重点工件包括数据流图、技术威胁建模报告、高层次的执行威胁建模报告、威胁清单以及威胁分析建议。

架构威胁分析

SDL 这一步的关键工件是概述威胁物化风险的工件。这一步需要的另一个工件的是威胁排名 / 优先级。

风险缓解计划

风险缓解计划列出了风险（和威胁），以及缓解、接受或容忍的方法。对于每个类别，它也概述了缓解风险的措施。最后，这份报告应在该项目的实际工作开始之前提交给企业用于签字。

政策符合性分析

这个工件是一个报告，它应该符合公司不同的安全和非安全策略，例如，待开发软件如何符合信息安全策略、数据治理策略、数据保留和加密技术政策等。

4.6.3 度量标准

以下指标应收集和记录在 SDL 周期的第 2 阶段。

- 业务威胁、技术威胁（映射到业务威胁）和威胁参与者的列表。
- 这个阶段之后不满足的安全目标个数。
- 遵守公司（现有）政策的百分比。
- 软件入口点的数量（使用的 DFD）。
- 风险（和威胁）接受、减轻和容忍的百分比。
- 重新定义的初始软件需求百分比。
- 产品中计划的软件体系结构（主要和次要）的变化数量。
- 根据安全要求所需的软件体系结构更改数量。

4.7 本章小结

SDL 模型中体系结构 (A2) 阶段的主要目标是从安全的角度确定软件的总体要求和结构。这个阶段的关键要素是从架构的角度来看的软件攻击面威胁建模以及元素说明文件；安全架构和设计准则的定义；持续的安全性、SDL、隐私政策，以及要求的合规审查；软件产品的安全性发布需求。

这些最佳实践将导致从安全的角度来看软件整体结构的定义。它们确定了其正常运行对安全性至关重要的组件，还确定了适用于软件产品架构（包括最低权限的应用程序）的相应安全设计技术，最小化软件产品的攻击面和任何基础设施。虽然较高层将会取决于较底层的服务，但是下层不能取决于较高层。虽然安全性体系结构确定了一个整体的安全性设计，但是该体系结构的各个元素的具体设计还是要按照个别的设计规格进行详细说明。

攻击面的各个独立元素的识别和测量给开发和软件安全团队提供了一个正在进行的安全度量，使它们能够检测该软件容易受到攻击的实例。在这个阶段，所有可以用来减少攻击面的异常必须进行审查，因为我们的目标是为正在开发的软件产品最大限度地提高安全性。威胁建模在组件对组件层面采用了结构化的方法来确定该软件必须管理的资产，以及通过哪些接口访问资产。在威胁建模期间任何捕获到可能对资产产生威胁的可能性都必须视作一种风险测量。用于降低风险的对策或补偿控制也在此阶段确定。在适当的和可行的环境下，应该利用工具捕获以机器可读形式存储和更新的威胁模型。在 SDL 的这一阶段同时还定义了具体的软件安全准则。

本章最后讨论了关键的成功要素、其重要性、该阶段的可交付成果以及应该在此阶段收集的指标。

参考文献

1. Kohnfelder, L., and Garg, P. (1999), "The threats to our products." *Microsoft Interface*, April 1, 1999. Retrieved from <http://blogs.msdn.com/sdl/attachment/9887486.Ashx>.
2. cisodesk.com (2012), "SiteXposure: Threat Modeling Process—Overview." Retrieved from <http://www.cisodesk.com/web-application-security/threat-modeling-overview>.
3. Hernan, S., et al. (2006), "Threat Modeling: Uncover Security Design Flaws Using the STRIDE Approach." *MSDN Magazine*. Retrieved from <http://msdn.microsoft.com/en-us/magazine/cc163519.aspx#S3>.
4. cisodesk.com (2012), "SiteXposure: Threat Modeling—Practice." Retrieved from

- <http://www.cisodesk.com/web-application-security/threat-modeling-in-practice>.
5. Hernan, S., et al. (2006), "Threat Modeling: Uncover Security Design Flaws Using the STRIDE Approach." *MSDN Magazine*. Retrieved from <http://msdn.microsoft.com/en-us/magazine/cc163519.aspx#S3>.
 6. [http://msdn.microsoft.com/en-us/library/ee823878\(v=cs.20\).aspx](http://msdn.microsoft.com/en-us/library/ee823878(v=cs.20).aspx).
 7. OWASP (2012), *Application Threat Modeling*. Retrieved from https://www.owasp.org/index.php/Application_Threat_Modeling#Data_Flow_Diagrams.
 8. Meier, J., et al. (June 2003), *Microsoft Corporation MSDN Library Doc: Improving Web Application Security: Threats and Countermeasures*. Retrieved from <http://msdn.microsoft.com/en-us/library/ff648644.aspx>.
 9. OWASP (2012), *Threat Risk Modeling*. Retrieved from https://www.owasp.org/index.php/Threat_Risk_Modeling.
 10. Ibid.
 11. Meier, J., et al. (June 2003), *Microsoft Corporation MSDN Library Doc: Improving Web Application Security: Threats and Countermeasures*. Retrieved from <http://msdn.microsoft.com/en-us/library/ff648644.aspx>.
 12. Microsoft MSDN (2012), *Cheat Sheet: Web Application Security Frame—Web Application Security Frame Categories*. Retrieved from <http://msdn.microsoft.com/en-us/library/ff649461.aspx>.
 13. OWASP (2012), *Application Threat Modeling*. https://www.owasp.org/index.php/Application_Threat_Modeling.
 14. Saitta, P., Larcom, B., and Eddington, M. (2005), *Trike v.1 Methodology Document [Draft]*. Retrieved from http://octotrike.org/papers/Trike_v1_Methodology_Document-draft.pdf.
 15. OWASP (2012), *Threat Risk Modeling*. Retrieved from https://www.owasp.org/index.php/Threat_Risk_Modeling.
 16. Saitta, P., Larcom, B., and Eddington, M. (2005), *Trike v.1 Methodology Document [Draft]*. Retrieved from http://octotrike.org/papers/Trike_v1_Methodology_Document-draft.pdf.
 17. U.S. Department of Homeland Security—US CERT (2009), *Requirements and Analysis for Secure Software—Software Assurance Pocket Guide Series: Development, Volume IV Version 1.0, October 5, 2009*. Retrieved from <https://buildsecurityin.us-cert.gov/swa/downloads/RequirementsMWV1001AM091111.pdf>.
 18. MyAppSecurity (2012), *Comparison of Threat Modeling Methodologies: P.A.S.T.A (Process for Attack Simulation and Threat Analysis)*. Retrieved from <http://www.myappsecurity.com/threat-modeling/comparison-threat-modeling-methodologies>.
 19. Morana, M., and Ucedavelez, T. (2011), "OWASP Threat Modeling of Banking Malware-Based Attacks Presentation," AppSec EU, June 10, 2011, Trinity College, Dublin, Ireland. Retrieved from https://www.owasp.org/images/5/5f/Marco_Morana_and_Tony_UV_-_Threat_Modeling_of_Banking_Malware.pdf.
 20. Morana, M. (2011), "Writing Secure Software Blog: Attack Simulation and Threat Analysis of Banking Malware-Based Attacks," June 10, 2011. Retrieved from <http://securesoftware.blogspot.com/2011/06/attack-simulation-and-threat-analysis.html>.
 21. MyApp Security (2012), *ThreatModeler*. Retrieved from <http://www.myappsecurity.com>.
 22. FiRST (2012), FiRST Homepage. Retrieved from <http://www.first.org>.
 23. FiRST (2012), "CVSS Frequently Asked Questions." Retrieved from <http://www.first.org/cvss/faq>.
 24. Software Engineering Institute—Carnegie Mellon (2012), *OCTAVE*. Retrieved from <http://www.cert.org/octave>.

25. OWASP (2012), *Threat Risk Modeling*. Retrieved from https://www.owasp.org/index.php/Threat_Risk_Modeling.
26. STANDARDS Australia–New Zealand (2012), *AS/NZS ISO 31000:2009 Risk Management—Principles and Guidelines*. Retrieved from <http://sherq.org/31000.pdf>.
27. ISO (2012), *ISO 31000:2009—Risk Management—Principles and Guidelines*. Retrieved from http://www.iso.org/iso/catalogue_detail?csnumber=43170.
28. Cisodesk (2012), *Threat Modeling—Practice Guide*. Retrieved from <http://www.cisodesk.com/web-application-security/threat-modeling-in-practice>.
29. OWASP (2012), *Application Threat Modeling: STRIDE Threat & Mitigation Techniques List*. Retrieved from https://www.owasp.org/index.php/Application_Threat_Modeling.
30. OWASP (2012), *Application Threat Modeling: ASF Threat & Countermeasures List*. Retrieved from https://www.owasp.org/index.php/Application_Threat_Modeling.
31. OWASP (2012), *Application Threat Modeling*. Retrieved from https://www.owasp.org/index.php/Application_Threat_Modeling.
32. Ibid.
33. Ibid.
34. Ibid.
35. Shostack, A. (2008), “Experiences Threat Modeling at Microsoft.” Retrieved from <http://www.homeport.org/~adam/modsec08/Shostack-ModSec08-Experiences-Threat-Modeling-At-Microsoft.pdf>.

设计和开发 (A3): SDL 活动与最佳实践

设计和开发 (A3) 阶段 (见图 5-1) 是软件最终用户最为关心的部分。在这个阶段你会做策略一致性分析, 创建测试计划文档, 更新威胁模型 (根据需要), 进行安全性设计分析和总结, 以及做隐私实施评估等, 帮助你安全地部署软件、建立开发最佳实践, 从而在开发阶段的早期消除安全问题和隐私问题。你将会在 SDL 的设计和开发 (A3) 以及发布 (A4) 阶段进行静态分析。下一章会提供一个关于静态分析的详细描述。你将能够构建一个计划来使你的项目能够通过 SDL 处理剩下部分, 包括实施、验证以及最终发布。在设计和开发 (A3) 过程中, 通过使用功能和设计规范, 你能够构建最佳的实践。

5.1 A3 策略一致性分析

如第 4 章所述, A3 的策略一致性分析是对 A2 策略一致性分析的延续。在这个阶段, 会对 SDL 域外部的所有策略进行评估。这是为了保证公司外部的开发机构在软件开发过程中遵守公司内部的安全和隐私需求以及开发指南。公司的安全和隐私策略会对需要达到的安全和隐私特征进行描述, 并指导设计人员和开发人员如何实现这些特征。其他策略可能针对公司内部或公司外部在软件中使用的第三方或开源软件的管理, 控制源代码以及其他智力成果。假如软件安全组是与集中的信息安全组是分开的, 那么两个组都要基于所有的原则和策略进行协作是很重要的, 包括开发、版本发布后的安全支持以及产品的反馈。同样重要的是在隐私功能上的合作, 无论是集中式组还是外部法律决策。

5.2 安全测试计划构成

测试活动用来验证产品发布后的安全措施是否能有效降低客户或恶意用户发现安全问题的可能。软件通过安全测试、工件 (artifact)、报告和工具对软件的安全性进行验证。我们的目的不是为了证明产品不安全, 而是在产品提供给客户前保证软件的健壮性和安全性。这些安全测试方法确实能发现安全 bug, 尤其是在那些没有经历关键的安全开发流程变更的产品中。安全测试和评估的结果也可能在安全控制中发现缺陷, 安全控制用于保护正在开发的软件。因此需要制定详细的行动计划和里程碑进度, 以记录计划的整改措施, 进而增强安全控制的有效性, 并在软件发布之前提供必要的安全措施。

正如第 4 章讨论的风险分析方法, 从整体上确保安全测试的有效性是很有必要的。安全测试通过设计和代码分析、软件行为调查确认该软件符合安全要求。换句话说, 安全测试是为证明软件功能被需求覆盖, 而每个需求都至少要有一个测试用例进行测试。需求总数和已经测试的需求数量可以通过测试用例和功能需求进行跟踪; 已测试需求占需求总量的比例将成为测试需求的指标。

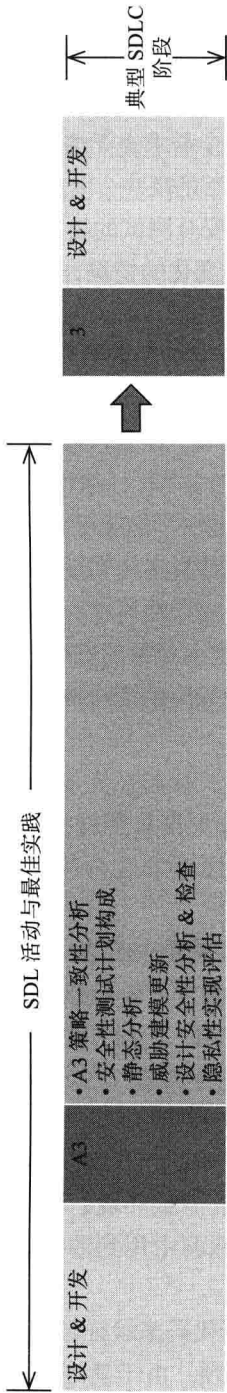


图 5-1 设计与开发 (A3): SDL 活动与最佳实践

安全测试的另一个要素是识别软件缺陷，以避免安全违规和不遵守安全要求的情况，从而导致软件出现故障或不符合软件安全性需求。正如 4.3.3 节所述，由于资源有限，安全测试计划往往集中在软件中关键的需求条目上。一个主测试计划用于勾勒整个测试过程的概况，并且为单项测试阶段和各个模块增加详细的测试计划。

虽然基于需求的传统测试对于验证软件实现的安全性功能的正确性和充分性是必要的，但是再多的测试也不能充分证明软件是没有漏洞的；任何这样的测试只能提供一部分观点以证明软件的安全性。即使稳定性最强的安全需求也无法涵盖现实世界中所有可能出现的情况，也无法观测到软件在异常条件或不良条件下的反应。

通常情况下，基于需求和风险的软件安全测试的缺点可以概括如下。

- 在开发中你遇到的软件威胁性质的变化的情况，可能会导致发布前和发布后出现新的攻击方法。
- 作为一个 SaaS 或者基于云的解决方案的功能组件，你的软件可能改变其物理位置，从而改变了攻击面，因此需要新的安全补丁或安全措施。
- 在当今竞争激烈和快速变化的环境中，在软件准备测试的时候，基于需求的设计假设可能已过时。
- 其他因素经常频繁改变，并有可能远快于你能够跟进的速度。
- 如果软件准备使用第三方组件，在项目实施初期规划的架构可能无法适用于未来的版本。
- 最初的设计无法考虑到由于设计变更而导致的安全漏洞，或者新的版本带来的漏洞。

这个列表强调，在本节中描述的基于风险的安全测试为什么要始终优于基于需求的传统测试。

正如前面提到的贯穿本书的主题所述，比起在完成编码，甚至更糟糕的情况——完成部署后——再去纠正安全问题，在软件生命周期早期就将安全“构建”进软件成本要低得多。这就是为什么要从 SDL/SDLC 过程的早期到软件生命周期的结束都要严格执行安全检查和测试的原因。

从软件安全测试的角度来看，测试者就是攻击者。构建对软件滥用和误用的测试场景，结合已知的攻击方法，发现技术和威胁模型中无效的部分。通过全面的测试场景和测试用例，就可以获得测试结果。可以肯定的是，在上一阶段开发的所有不合适的测试用例会被执行和全面测试。如果在每个迭代开发阶段都进行测试，在 SDL 的早期就会发现大量的设计缺陷，这个阶段发现的问题将会占整个阶段发现问题的 95%。^[1]

测试环境只是与部署环境应尽可能一致，并且完全隔离。还应该对配置进行严格的管理和控制，防止测试数据、测试工具、集成测试环境、测试计划、原始和最终的测试结果被破坏。同样重要的是，要确保每个软件漏洞中用到的工具集和测试技术都适用于当前的被测试系统。

安全性测试和功能性测试都需要执行代码来验证结果。测试也并不总是自动的，通常需要经验丰富的软件安全架构师进行人工干预。由于某些软件生态系统内部的复杂性和交互性，比如 SaaS 或云环境，因此需要专家确认测试方法是否能够适用于更大的范围。

正如前面所提到的，测试计划列出了需要测试的功能，需要保护的对象，以及应用程序将如何应对特定的攻击。测试计划是项目管理团队、开发团队和安全团队共同努力的结果，

除此之外，还要描述测试计划的逻辑流程，包括谁来执行测试以及测试的开始时间和结束时间。

下面列出了实现测试计划的通用步骤（无论使用什么策略、框架或标准）。

- **定义测试脚本。**脚本是非常详细且包括测试逻辑步骤的指令集，用于告诉测试人员或测试工具在测试期间做什么。功能测试脚本是一步一步的指令，描绘了一个特定的场景或情景，包括将会遇到的情况和预期的结果。安全测试脚本是专门用于测试应用程序安全性的脚本。这些脚本的依据来源于在设计阶段生成的威胁模型。误用用例定义那些需要保护（资源），什么类型的攻击可以访问这些资源。安全测试脚本定义了执行这些攻击的行为。
- **定义用户社区。**定义用户社区帮助测试人员识别错误和风险的可接受程度。
- **识别项目障碍物。**用例中需要定义必备的和“如果可用”的情景。如果没有定义，就需要重新审视测试需求，从而使这些规范文档化。
- **识别内部资源。**内部资源来自于公司的组织，包括开发商、分析师、软件工具，有时还可能项目经理。
- **识别外部资源。**外部资源是为了测试应用系统和提交报告而临时加入项目的人员或工具。外部资源最适合安全测试，因为他们一般都在安全编程技术方面受过严格的训练，同时他们远离代码和任何内部策略。如果需要外部资源，测试计划需要回答以下问题：（1）他们测试什么？（2）他们的报告提交给谁？（3）他们与谁一起工作？^[2]

我们将软件的安全属性和行为划分为外部实体的和内部实体的：外部实体包括用户、环境和其他软件；内部实体包括软件自身有交互行为并且作为主要测试对象的内部组件。具体来说，应该验证软件以下的属性和行为：

- 它的行为是可预测和安全的。
- 它不暴露漏洞或者缺陷。
- 它的错误和异常处理程序，使得它能够在面对攻击或者故意故障的时候，维持安全状态。
- 它满足了所有的规定和隐含的非功能性的安全要求。
- 它并没有违背任何规定和隐含的非功能性的安全要求。
- 它并没有违反任何规定的安全限制。
- 使用模糊或混淆技术对尽量多的源代码进行处理，以防止攻击者使用逆向工程对程序进行攻击。^[3,4]

安全测试计划应该包含在整个软件测试计划中，并定义所有与安全相关的测试活动，包括如下：

- 安全测试用例或场景（基于滥用用例）
- 测试数据，包括攻击模式
- 测试语言（如果有一个被使用）
- 测试工具（白盒和黑盒、静态和动态）。
- 测试分析，即对各种测试工具的测试结果进行解释和关联。^[5,6]

软件安全测试技术可以分类为白盒测试、灰盒测试或者黑盒测试三类。

- **白盒。**从内部的角度测试软件，即充分了解软件内部；源代码、架构和设计文档，以

及配置文件可用于分析。

- **灰盒**。设计测试用例的目的是分析源代码，同时也使用黑盒测试技术；源代码和可执行二进制文件都可用于分析。
- **黑盒**。从外部的角度测试软件，没有关于软件的任何知识；只有二进制可执行文件或中间字节码可用于分析。^[7,8]

上述常用的安全测试技术可以按如下分类。

- **源代码分析（白盒测试）**。源代码分析是对软件的源代码进行检测，用来发现源代码中可能导致安全漏洞的部分。相对于其他类型的静态分析工具，源代码安全分析器的主要优点是源代码的可用性。比起从字节码或者二进制中逆向工程得到的代码，源代码包含更详细的信息，因此，更容易发现可导致安全漏洞的软件缺陷。此外，如果进行分析的是原始的源代码，会更容易修复发现的安全漏洞。
- **基于属性（白盒测试）**。基于属性的测试是由美国加州大学戴维斯分校（University of California Davis）创立的正式的分析技术。基于属性的测试验证了该软件的功能是否满足其设计规格。这种方法通过检查源代码中与安全相关的属性实现，例如缺乏不安全的状态转换。然后将代码中与安全相关的属性与软件的设计规格进行比对，确定预期的安全假设是否得到满足。
- **源代码错误注入（白盒测试、灰盒测试）**。错误注入技术是通过测试所有代码路径以提高代码覆盖率的技术。尤其是错误处理代码路径，这部分代码在功能测试中可能不会被测试到。在错误注入测试中，错误被注入软件中，用来模拟针对软件或软件运行环境的攻击。在源代码错误注入测试中，测试者需要确定在什么情况下触发环境错误。之后在源代码中插入能够导致这些错误的环境数据。注入错误后的源代码会被编译和执行，当错误注入代码部分执行时，测试者观察软件状态是否发生变化。可以让测试者通过改变环境观察软件安全状态和不安全状态的变化。测试者还可以分析那些由注入错误传播后导致的错误。这种类似于错误传播分析的分析方法包括两种典型的技术：扩展传播分析和接口传播分析。
- **动态代码分析（灰盒测试）**。动态代码分析在应用程序运行时对代码进行检测，测试者可以通过跟踪外部接口以及和外部接口进行交互的代码，找到测试中发现的漏洞和异常的源代码并进行修复。与静态分析不同的是，动态分析允许测试人员修改软件配置和环境行为相关的组件来发现漏洞。由于软件自身的配置与真实的测试环境并不能完全对应，因此一般采用测试工具对测试交互、关联输入、环境条件以模拟的方式进行测试。
- **二进制错误注入（灰盒测试、黑盒测试）**。二进制错误注入是一种运行时分析技术，在错误注入时监控应用程序。通过监视系统调用，测试者能够识别系统调用的名称、每一个调用的参数和调用的返回码。这允许测试者发现被调用软件访问的资源名称和类型、资源是如何使用的以及每一次访问尝试是成功还是失败。在二进制错误分析中，错误被注入应用程序的环境资源中。
- **模糊测试（黑盒测试）**。模糊是一种用于检测软件中错误和安全相关 bug 的技术，通过向程序中提供随机输入实现。与静态分析中源代码逐行评审以发现 bug 的方式相反，模糊测试通过给程序产生各种有效和无效的输入并监控结果来进行分析，这种测试有

可能导致程序崩溃。

- **二进制代码分析 (黑盒测试)**。二进制代码扫描器分析机器码来建立语言无关的表示模型, 包括程序行为、控制和数据流、调用树和外部函数调用。这样的模型会被自动漏洞扫描工具遍历, 从而找出常见的代码错误和简单后门引起的漏洞。源代码分析工具可以基于该模型生成用来表示程序行为的可读代码, 人工代码检查可以根据这一结果发现设计阶段产生的安全缺陷以及无法用自动扫描器发现的后门。
- **字节码分析 (黑盒测试)**。字节码扫描器与源代码安全分析器相似, 它是通过检测字节码来发现漏洞的。例如, Java 语言编译成一个与平台无关的字节码格式, 其在运行时环境 (Java 虚拟机) 中执行。包含在原始 Java 源代码中的大部分信息保留在已编译的字节码中, 因此使得反编译成为可能。可以在不能得到软件源代码的测试中使用字节码扫描器——例如用来评估第三方软件组件对应用软件安全性的影响。
- **黑盒调试 (黑盒测试)**。低级语言, 如 C 语言或汇编语言的调试器, 允许测试者监控程序的执行、启动和停止程序、设置断点和修改变量值。调试器只用在源代码或编译符号可用时才能够使用。源代码和编译符号可以跟踪内部变量的值, 从而发现程序内部行为的特点。然而, 有的测试只能对编译器生成的没有设置编译符号或调试标志的二进制代码进行测试, 如商业软件、旧式软件, 以及为防止逆向工程使用代码混淆技术处理过的软件。在这种情况下, 传统的调试方法是无法使用的。应当指出的是, 如果调试工作的重点是在与外部组件的软件交互上, 只需要二进制代码就可以了。
- **漏洞扫描 (黑盒测试)**。针对操作系统和应用程序的自动化漏洞扫描工具有商用的和开源的两类, 它们通过使用攻击模式对被测目标进行测试以验证是否包含已知的漏洞。就像病毒扫描工具依赖特征知识库识别病毒一样, 漏洞扫描工具依赖将漏洞特征做成“签名”的知识库识别漏洞。就像自动代码审查工具一样, 尽管许多漏洞扫描工具尝试使用一些机制自动识别漏洞, 但仍然无法检测复杂的漏洞或没有保存在知识库中的漏洞。除了基于漏洞签名知识库的漏洞扫描工具外, 大多数漏洞扫描工具采用模拟攻击者的攻击方式来探测软件是否可以暴露漏洞。
- **渗透测试 (黑盒测试)**。安全性测试的一部分, 在该测试中评价者试图绕过系统的安全功能。评价者假装可以使用所有系统的设计和实现文档, 其中可以包括系统源代码列表、手册和系统流程图。评价者和普通用户在相同的条件下工作。^[9-12]

测试计划用来组织安全测试流程, 描述了会对哪些软件组件进行测试以及每个测试的流程是什么。测试计划并不单单是一个要完成的任务列表; 它还要包括对软件的哪些方面进行测试, 使用了哪些技术进行测试, 以及每种测试方法的描述, 包括先决条件、配置、执行以及需要从测试结果中获得哪些结论。由于时间和预算的限制, 往往无法对软件所有的组件都进行测试, 因此要在测试前就要做风险分析。安全测试计划是一个持续的过程, 随着测试信息的增多, 测试的细节会不断补充进来。

在开发部门修复问题后, 会将程序送回测试部门进行回归测试。由于在软件全部开发完成后再进行测试是很低效的, 因此通常采用的方式是软件的一部分组件正在开发, 一部分组件正在进行测试。对于大型项目来说, 测试计划通常分解成不同的测试周期。对于上面给出的例子, 为了对软件进行复测就需要创建一个测试周期; 当遇到由于开发工作性质的原因导致不同的模块在不同的时间进行测试的情况, 也需要创建测试周期。

为了将重复测试的数量限制到最小，需要考虑组件间的依赖关系，以及 SaaS 或云环境中的元素和软件组件间的关系。开发部门往往有着严格的开发流程，因此安全组需要在 SDLC 过程的开始阶段就参与进来。安全团队需要尽早指定软件组件的测试顺序，确保测试到一个组件时，它所依赖的组件已经完成了开发与测试。“构建安全”的概念不单单用来构建安全计划，必须严格遵守这一计划。这意味着安全团队需要包含在一般的测试计划讨论中，以确保安全元素被包含在测试环境的验证和测试数据中，以及测试用例如何定义测试条件。测试条件是真正要进行测试的，以观察该软件实际上如何做出回应。测试用例在测试执行过程中创建，并包括有关测试前置条件和后置条件的信息，如何设置和终止，以及如何评价测试结果。

自动化是让测试流程平稳进行的关键，最重要的是，测试的可重复性。正如前面所提到的，并不是所有测试都能满足这些要求，人工参与是必要的，特别是经验丰富的软件安全架构师。测试计划要尽量充分，让测试人员了解在每个测试中需要得到的是什麼，以及需要做哪些准备。

Cigital 公司的 Michael 和 Radosevich 在 2005 年名为《基于风险和功能的安全测试》(Risk-Based and Functional Security Testing) 的白皮书中，列出了以下关于安全测试的典型元素，可以用来作为制定测试计划的准则。

- 目标
- 在测软件概述
 - 软件和组件
 - 测试边界
 - 测试限制
- 风险分析
 - 风险分析概要
- 测试策略
 - 假设
 - 测试方法
 - 不需要测试的项
- 测试要求
 - 功能测试要求
 - 安全测试要求
 - 安装 / 配置测试要求
 - 压力和负载测试要求
 - 用户文档
 - 人员要求
 - 设备和硬件要求
- 测试环境
- 测试用例规格
 - 唯一的测试标识符
 - 要求可追溯性 (需求文档中有多少需求数量参与测试用例验证)
 - 输入规格

- 输出规格 / 预期结果
- 环境需求
- 特定程序要求
- 测试用例依赖
- 测试自动化和测试件
 - 测试件架构
 - 测试工具
 - 测试件
- 测试执行
 - 测试准入条件
 - QA 验收测试
 - 回归测试
 - 测试程序、特殊要求、程序步骤
 - 测试计划
 - 测试准出条件
- 测试管理计划
- 定义和缩略词^[13]

5.3 威胁模型更新

在通过第 4 章介绍威胁模型构建过程之后，知道构建过程何时完毕是很重要的。需要回答以下几个问题，答案可能取决于商业竞争与安全风险之间的取舍。

1. 你开发的软件是否能符合所有相关政策、法规、条例的规定？并且对于每个需求获得各个层次的批准？
2. 所有的利益相关方是否对威胁建模过程中识别出的安全和风险进行了检查？相关的架构师、开发人员、测试人员、项目经理以及其他所有了解软件的人员都应成为威胁模型构建做出贡献并进行核查。为了保证威胁模型覆盖全面，应广泛征求各方意见。所有相关人员在威胁和风险的认知上达成一致是很重要的。否则，针对威胁模型实现相关措施的落实会面临很多困难。
3. 你是否就建立威胁模型和针对威胁模型采取的应对措施、测试所需的时间和资源与相关方达成一致？
4. 你对风险和威胁的排名是否与相关人员达成了共识？如果你是一个软件购买者，是否会同意这个排名？

5.4 设计安全性分析和检查

在 1974 年的一篇文章中，来自弗吉尼亚大学的 Saltzer 和 Schroeder，提供了通过保护存储信息所需的软件和硬件来保护信息的方法。^[14] 文章提出了以下 11 个安全设计原则。

1. **最小权限。**最小权限原则主张，对于完成一项任务的个人、进程或其他实体，只在可完成该任务的最短时间内分配给该实体能够完成该任务的最小权限和资源。这种方法避免了未经授权访问敏感信息的情况。
2. **职责分离。**该原则要求，必须满足多个条件，才能完成指定的敏感行为或接触敏感信

息。系统通过将实体的权限进行分割来满足该原则。

3. **纵深防御。**通过多层次的保护以确保当一层保护被破坏后，后面各层能起到保护应用的作用。

4. **故障安全。**这意味着如果一个系统出现故障，它依然能处在安全并且数据不受到损害的状态。如果系统无法从故障状态自动恢复，应该只有系统管理员能进入系统，普通用户直到系统恢复正常后才能进入系统。

5. **经济机制。**该机制促进了简单易懂的保护机制的设计与实现，消除了意想不到的访问路径或者可以很容易地确定与消除它们。

6. **完全仲裁。**计算机系统中所有主体对客体的访问都必须遵循一个有效且高效的授权过程。这种仲裁不能暂停或能够被绕过，即使信息系统初始化后，关机，重新启动，或是在维护模式下。完整的仲裁包括：(a) 识别访问请求的实体；(b) 验证请求是否未被修改；(c) 使用适当的授权流程；(d) 复审预先授权同一个实体的请求。

7. **开放式的设计。**我们一直在讨论，比起将源代码闭源，将代码开源并在社区以更大的范围对软件进行评估的优点和好处。以访问控制系统为例，在大多数情况下，比起一个闭源的访问控制系统，一个开放的访问控制系统的设计会被更大范围、更多的专家进行评估，从而获得更加安全的身份验证方法。

8. **最小公共机制。**这一原则是说，多个用户间共享的保护机制数量应该尽可能小，例如：共享访问路径可能会导致未授权的信息交换。这种会导致未预期的数据传输的共享访问路径(叫作转换通道)。为了满足最小公共机制原则，应尽量减少公共的安全机制。

9. **心理可接受性。**指与访问控制机制交互的用户接口的易用性和直观性。用户必须能够理解用户界面，使用该界面时无需复杂的提示。

10. **最薄弱环节。**古语道，“一条链子的强度取决于其最薄弱的环节。”信息系统安全性所能达到的强度等同于其最薄弱环节所能达到的安全性。因此一个很重要的工作是，识别安全防护环节和安全层面中最薄弱的部分，有针对性地进行防御和提升，使系统存在的风险降低到可接受的程度。

11. **利用现有的组件。**在很多情况下，一个信息系统的安全机制并未合理配置或者发挥其最大功效。审查信息系统现有的安全机制和设置，使它们达到最佳的安全设计预期很可能会大大提升信息系统的安全状况。另一个提高系统安全性的方法是，利用现有组件将系统划分为一个个防御单元。这样，如果攻击者绕过了一个防御单元的防御机制，将不会对其他防御单元造成破坏，使计算资源遭受的破坏降低到最小。^[15,16]

设计好的软件并不容易，使它安全就更困难了。虽然从用户角度上来讲一个软件的缺点并不会带来很大的问题，但是从安全角度来讲则不同，因为攻击者可以进行针对性的测试提高触发软件缺陷的几率。有的缺陷因为随机出现或出现概率很低而被忽略，但是攻击者很可能利用这样的缺陷对应用软件造成很大的影响。在总结 Saltzer 和 Schroeder 的设计原则时，虽然原则内容很明确，但是缺乏满足原则的标准。幸运的是，我们了解信息安全的概念，能够将公认的保密性、完整性、可用性的属性融合到设计原则中，避免因设计原则的把握不当而带来的安全隐患。但然，从不同的角度对安全的认识也是不同的，例如一个软件开发人员可能认为安全主要集中在软件质量上；一个网络管理员认为安全主要集中在防火墙、IDS/IPS 系统、事件响应和系统管理方面；甚至那些管理者和学者，他们可能认为安全主要体现在上

述的经典设计原则或者多种多样的安全模型上。这些观点在构建安全系统和相关的威胁模型过程中是很重要的。

详细设计工作用来有效地构建满足用户需求的软件组件。对每个软件组件可能存在的漏洞进行深入分析,包括每个组件的特征、属性和服务。通过第4章描述的对每个软件组件在用例和误用例情况下结果的分析,开发者可以设计出更加合适、清晰的处理方案,整个团队更加直观地了解软件安全机制是如何控制应用软件的。这个过程可以将用例转换为实际的软件需求规格说明书。开发者检查他们自己开发的特定应用软件时,也应该检查软件运行的环境是否存在漏洞,包括网络、架构以及依赖的其他软件。开发安全团队一项重要的工作是跟踪有可能影响到软件及其运行环境的漏洞——包括发布前的和发布后的——从而避免已知和未知攻击手段从软件内部或外部进行攻击。比起在软件发布后再进行处理,在设计阶段和开发阶段就识别风险并采取相应的处理措施要更加简单和节约成本。代码安全设计是最为关键的,需要投入大量资源,但还是无法避免错误,新的攻击方法会出现,因此直到软件生命周期的结束都需要持续研究和评估新的新的攻击方法和漏洞。这就是第4章介绍的最小化攻击面的重要性,结合良好的设计原则可以最大程度降低代码审查环节遗漏的安全缺陷的严重性。作为安全工作的组织框架,设计阶段可以策略性地将问题切分成一个个小问题以便解决。威胁建模对于这次尝试过程的成功很重要,因为它通常会识别一个直到很晚都不会注意的安全设计问题。使用数据流图以及对攻击方法进行头脑风暴并审核已知的检查列表进行下一步的工作。尽早使用最适合你的方法并进行必要的研究以完善你的设计阶段,从而使开发阶段出现的组件故障最小化。

5.5 隐私实现评估

笔者认为,最简洁、清晰并且经过现场检验的软件开发隐私措施评估现场测试流程、指南来自微软的三篇重要的文档。

1. 开发软件产品和服务的微软隐私指南,版本 3.1;2008 年 9 月 (*Microsoft's Privacy Guidelines for Developing Software Products and Services, Version 3.1; September 2008*)^[17]

2. 微软 MSDN 的 SDL——过程指南——附录 C:SDL 隐私问卷调查 (*Microsoft MSDN's SDL—Process Guidance—Appendix C: SDL Privacy Questionnaire*)^[18]

3. 微软 SDL 的简单实现 (*Microsoft's Simplified Implementation of the Microsoft SDL*)^[19]

要确定隐私影响等级以及评估相关工作的流程和指南,请参考“开发软件产品和服务的微软隐私指南”,版本 3.1;2008 年 9 月^[20]和“微软 MSDN 的 SDL——过程指南——附录 C:SDL 隐私问卷调查”^[21]。评级(P1、P2 或 P3)从隐私的角度表现了软件的风险等级。你只需要按照指南的步骤就可以完成评估。

- **P1: 高级隐私风险。**特征、产品或服务对个人身份鉴别信息 (personally identifiable information, PII) 进行存储或传输,更改相关的配置或文件类型,或安装软件。
- **P2: 中级隐私风险。**影响特征、产品或服务的隐私的独立行为是一次性的、用户发起的、匿名数据传输的(如,用户点击一个链接软件就跳转到一个外部网站)。
- **P3: 低级隐私风险。**特征、产品或服务内部没有影响隐私的行为。不传输匿名或个人数据,不存储 PII,没有配置改变用户的利益,不安装软件。^[22]

这些问题是为了完成 SDL 的隐私方面的检查而设计的,你可以独立完成几个部分,例如

初步评估和详细评估。建议你与你的隐私顾问一起完成其他部分，如隐私检查。^[23]

保护用户隐私最好的方法之一是不收集用户数据。架构师、开发者、数据收集系统管理员应该不断问自己：

- “我需要收集这个数据吗？”
- “我有一个合理的商业目的吗？”
- “客户会支持我的商业目的吗？”^[24]

开发组织必须牢记，客户可以控制自己的个人信息，他们需要知道哪些个人信息会被收集，会分享给谁，以及如何被使用。此外：

- 从客户电脑获取他们的个人信息之前必须征得他们同意；
- 如果客户用户的个人信息是通过互联网传输的并远程存储，必须为他们提供访问和修改个人信息的方法。^[25]

“开发软件产品和服务的微软隐私指南”，版本 3.1^[26] 的编写过程参考了大量文献，包括：经济合作与发展组织（Organization for Economic Co-operation and Development, OECD）公平信息惯例^[27] 的核心概念和隐私法律，比如欧盟数据保护法^[28]，美国 1998 年的儿童在线隐私保护法（Children’s Online Privacy Protection Act, COPPA）^[29] 以及美国计算机欺诈与滥用法案^[30] 及其修正案。“开发软件产品和服务的微软隐私指南”文档分为两个主要部分。第 1 部分介绍关键的隐私概念和定义。第 2 部分列举了面向特定软件产品和网站开发场景的详细指南。“开发软件产品和服务的隐私指南”的目录表示如下。

Table of Contents	
	Introduction
1	Basic Concepts and Definitions
1.1	User Data
1.1.1	User Data Categories
1.2	Data Minimization and Data Management
1.2.1	Minimizing Data Collected
1.2.2	Limiting Access to “Need to Know”
1.2.3	Reducing the Sensitivity of Data Retained
1.2.4	Reducing Duration of Data Retention
1.3	Notice, Choice, and Consent
1.3.1	Types of Notice
1.3.2	Types of Consent
1.4	Notice Mechanisms
1.4.1	Just-in-Time Notice
1.4.2	First Run Notice
1.4.3	Installation Time Notice
1.4.4	“Out of the Box” Notice
1.5	Security
1.6	Access
1.7	Data Integrity
1.8	Types of Privacy Controls
1.8.1	User Controls
1.8.2	Administrator Privacy Controls
1.9	Shared Computers
1.10	Children’s Privacy
1.11	Software Installation

1.12	Server Products
	Third Parties
1.13	Web Sites and Web Services
1.13.1	Using P3P for Privacy Statements
1.13.2	Using Cookies
1.14	Special Considerations
1.14.1	Pre-Release Products
1.14.2	Essential Transfers and Updates
1.14.3	File and Path Names
1.14.4	IP Address
1.14.5	When Things Change
2	Guidelines
2.1	How to Use This Section
2.2	Scenarios
	Scenario 1: Transferring PII to and from the Customer's System
	Scenario 2: Storing PII on the Customer's System
	Scenario 3: Transferring Anonymous Data from the Customer's System
	Scenario 4: Installing Software on a Customer's System
	Scenario 5: Deploying a Web Site
	Scenario 6: Storing and Processing User Data at the Company
	Scenario 7: Transferring User Data outside the Company
	Scenario 8: Interacting with Children
	Scenario 9: Server Deployment
Appendix A	Windows Media Player 10 Privacy Experience
Appendix B	Security and Data Protection
Appendix C	User Data Examples

作为设计要求活动的一部分，额外的隐私操作包括：创建隐私设计规范、规范审核、最小加密设计要求的规范。设计规范应该说明直接暴露给用户的隐私功能，比如那些需要用户验证才能访问的特定数据，或者用户同意后才能使用的高风险的隐私功能。此外，所有的设计规范应该说明如何安全地实现给定的特征或功能。通过设计规范验证应用程序的功能规范是个很好的方法。功能规范应该包括：（1）特征或功能的准确、完整说明；（2）使应用程序达到安全的状态所需的特征或功能配置。^[32]

为了保护 PII 数据的安全控制，必须考虑数据保护的各个方面，包括（但不限于）：访问控制、数据传输和存储加密、物理安全、灾难恢复和审计。很多情况下，保护关键业务数据使用的安全机制，同样可以用于保护客户和员工的个人信息。关键业务数据的保护来自于妥协和损失信息的机密性和专有性。根据本节介绍的指南，在对机构收集、存储、传输 PII 的行为进行识别、理解、分类后这才能确定。

5.6 成功的关键因素和度量标准

5.6.1 成功的关键因素

SDL 第 3 阶段的成功取决于安全测试计划、设计安全分析审查和隐私实施评估。在这个

阶段中，创建了 SDL 剩下的所有流程的计划，从执行到测试。表 5-1 列出了第 3 阶段的关键成功因素。

表 5-1 关键成功因素

关键成功因素	描 述
1. 全面的安全测试计划	映射 SDLC 不同阶段需要的安全测试类型
2. 有效的威胁建模	识别软件威胁
3. 设计安全分析	不同软件组件的威胁分析
4. 隐私实施评估	根据评估实施隐私相关控制所需要的工作量
5. 策略一致性审查（更新）	涉及阶段 3 的策略遵从更新

成功因素 1：全面的安全测试计划

这个阶段，安全架构师和评估小组定义了需要对软件进行测试的各个方面，以及版本发布前和发布后需要进行的测试类型及时间安排。安全测试计划应该将代码开发阶段与测试类型进行映射。例如，当软件进入静态分析阶段时，必须对最终提交的代码进行全面的静态分析。当软件进入预发布阶段时，应该进行安全漏洞评估、灰盒测试和二进制测试。这个阶段与整个 SDL 中最关键的成功因素之一，就是安全测试计划能够在产品部署之前消除大多数的安全漏洞。

成功因素 2：有效的威胁建模

如果在这个阶段确定了新的威胁和攻击向量，需要对威胁模型和标准进行更新，以确保风险处理计划是全面的。

成功因素 3：设计安全性分析

与安全测试计划的准确性一样，从安全角度对软件设计的评审也许是在 SDL 前 3 阶段中最重要的成功因素。重要的是：最小化攻击面，完善设计原则以最大限度地降低安全漏洞的严重程度。

成功因素 4：隐私实施评估

对公司内部和外部隐私策略的遵守和实践的现状要尽可能准确。这将显著节约成本。例如，如果隐私实践要求 PII 数据全线加密，在设计阶段识别该需求是至关重要的。一旦该软件处在执行和发布阶段这就要付出很大的成本才能实现。笔者看到，“财富 500 强”企业是在服务包发布阶段做这项工作的。然而，很难准确地解决问题。它也是代价高昂的。这个问题的另一个例子是网络分段。在云 /Saas 类型的环境中，这是一个重要的决策。通常会有多个产品脱离共享环境。如何将它们分别进行有效的保护在设计阶段是很重要的问题。

成功因素 5：策略一致性审查（更新）

如果对现有的策略进行了更新，或确定补充了新的策略，需要审查需求是否满足新的策略。另一个例子是，更新策略包含了隐私相关策略或前瞻性战略。

5.6.2 可交付成果

表 5-2 列出了 SDL 这个阶段的可交付成果。

更新的威胁建模工件

需要创建更新的数据流图、技术威胁建模报告、高级执行威胁报告、威胁列表，以及基

于攻击向量的新需求 / 输入的威胁分析的建议。

设计安全审查

这是一份正式的规范，基于安全架构和安全评估的软件设计审查列出了软件组件的更改。

安全测试计划

这是一个正式的安全测试时间表，将 SDL 过程中不同的阶段与不同类型的安全测试进行映射（静态分析、模糊测试、漏洞评估、二进制测试等）。

更新的策略遵从分析

策略遵从分析标准（见第 4 章）应该基于任何新的要求或策略更新，这些新的需求或策略可能已经在 SDL 的这个阶段出现。

隐私实施评估结果

基于隐私风险（高、中、低）建立的用来实现隐私实施评估建议的路线图。

表 5-2 A3 阶段可交付成果

可交付成果	目 标
更新的威胁建模标准	数据流图、要素、威胁列表
设计安全审查	基于安全评估的软件组件设计修改
安全测试计划	计划减轻、接受或容忍风险
更新的策略遵从分析	坚持遵守公司政策
隐私实施评估结果	隐私评估的建议

5.6.3 度量标准

由于一些成功的隐私和可交付成果与 SDL 的第 2 阶段相似，相同的度量指标应该收集和记录：

- 威胁、可能性和严重性
- 符合公司政策的百分比（更新）
- 符合第 2 阶段和第 3 阶段的百分比
- 软件的切入点（使用 DFD）
- 接受的风险相对于缓解的风险的百分比
- 最初的软件重新定义要求的百分比
- 软件架构变化的百分比
- 没有响应的软件安全测试的 SDLC 阶段的百分比
- 与隐私控制有关实现的软件组件的百分比
- 代码行数
- 使用静态分析工具发现的安全缺陷数量
- 使用静态分析工具发现的高风险缺陷数
- 缺陷密度（每 1000 行代码中的安全问题）

需要注意的是，如果有太多涉及隐私的控制需要在软件组件中实现，你可能要审查组件的设计。

5.7 本章小结

在我们关于设计和开发的讨论中（A3 阶段），我们描述了以下几个方面的重要性：策略一致性分析、测试计划文档创建、威胁建模更新、设计安全分析和审查完成，以及隐私实施评估。除了这些之外，最佳实践根据功能和设计规范实施，用于整个 SDL 过程的剩余阶段。本章最后讨论了这一阶段的关键成功因素、可提交的成果和度量指标。

参考文献

1. McConnell, S. (1996), *Rapid Development*. Microsoft Press, Redmond, WA.
2. Grembi, J. (2008), *Secure Software Development: A Security Programmer's Guide*. Course Technology, Boston, MA.
3. Krutz, R., and Fry, A. (2009), *The CSSLP Prep Guide: Mastering the Certified Secure Software Lifecycle Professional*. Wiley, Indianapolis, IN.
4. Information Assurance Technology Analysis Center (ITAC)/Data and Analysis Center for Software (DACS) (2007), *Software Security Assurance State-of-the-Art Report (SOAR)*. Available at <http://iac.dtic.mil/csiac/download/security.pdf>.
5. Krutz, R., and Fry, A. (2009), *The CSSLP Prep Guide: Mastering the Certified Secure Software Lifecycle Professional*. Wiley, Indianapolis, IN.
6. Information Assurance Technology Analysis Center (ITAC)/Data and Analysis Center for Software (DACS) (2007), *Software Security Assurance State-of-the-Art Report (SOAR)*. Available at <http://iac.dtic.mil/csiac/download/security.pdf>.
7. Krutz, R., and Fry, A. (2009), *The CSSLP Prep Guide: Mastering the Certified Secure Software Lifecycle Professional*. Wiley, Indianapolis, IN.
8. Information Assurance Technology Analysis Center (ITAC)/Data and Analysis Center for Software (DACS) (2007), *Software Security Assurance State-of-the-Art Report (SOAR)*. Available at <http://iac.dtic.mil/csiac/download/security.pdf>.
9. Krutz, R., and Fry, A. (2009), *The CSSLP Prep Guide: Mastering the Certified Secure Software Lifecycle Professional*. Wiley, Indianapolis, IN.
10. Information Assurance Technology Analysis Center (ITAC)/Data and Analysis Center for Software (DACS) (2007), *Software Security Assurance State-of-the-Art Report (SOAR)*. Available at <http://iac.dtic.mil/csiac/download/security.pdf>.
11. Fink, G., and Bishop, M. (1997), "Property-Based Testing: A New Approach to Testing for Assurance." *SIGSOFT Software Engineering Notes*, vol. 22, no. 4, pp. 74–80.
12. Goertzel, K., et al. (2008), *Enhancing the Development Life Cycle to Produce Secure Software. Version 2.0*. U.S. Department of Defense Data and Analysis Center for Software, Rome, NY.
13. Michael, C., and Radosevich, W. (2005), "Risk-Based and Functional Security Testing." Digital white paper, U.S. Department of Homeland Security. Updated 2009-07-23 by Ken van Wyk. Available at https://buildsecurityin.us-cert.gov/bsi/articles/best-practices/testing/255-BSI.html#dsy255-BSI_sstest.
14. Saltzer, J., and Schroeder, M. (1974), "The Protection of Information in Computer Systems." Fourth ACM Symposium on Operating Systems Principle, October 1974.
15. Ibid.
16. Grembi, J. (2008), *Secure Software Development: A Security Programmer's Guide*. Course Technology, Boston, MA.
17. Microsoft Corporation (2008), *Privacy Guidelines for Developing Software Products*

- and Services, Version 3.1; September 2008. Available at <http://www.microsoft.com/en-us/download/details.aspx?id=16048>.
18. Microsoft Corporation (2012). MSDN, *SDL—Process Guidance—Appendix C: SDL Privacy Questionnaire*. Available at <http://msdn.microsoft.com/en-us/library/cc307393.aspx>.
 19. Microsoft (2011), *Simplified Implementation of the Microsoft SDL*. Available at <http://www.microsoft.com/en-us/download/details.aspx?id=12379>.
 20. Microsoft Corporation (2008), *Privacy Guidelines for Developing Software Products and Services*, Version 3.1; September 2008. Available at <http://www.microsoft.com/en-us/download/details.aspx?id=16048>.
 21. Microsoft Corporation (2012), MSDN, *SDL—Process Guidance—Appendix C: SDL Privacy Questionnaire*. Available at <http://msdn.microsoft.com/en-us/library/cc307393.aspx>.
 22. Microsoft (2011), *Simplified Implementation of the Microsoft SDL*. Available at <http://www.microsoft.com/en-us/download/details.aspx?id=12379>.
 23. Microsoft Corporation (2012), MSDN, *SDL—Process Guidance—Appendix C: SDL Privacy Questionnaire*. Available at <http://msdn.microsoft.com/en-us/library/cc307393.aspx>.
 24. Microsoft Corporation (2008), *Privacy Guidelines for Developing Software Products and Services*, Version 3.1; September 2008. Available at <http://www.microsoft.com/en-us/download/details.aspx?id=16048>.
 25. Ibid.
 26. Ibid.
 27. Organisation for Economic Co-operation and Development (1980), *OECD Guidelines on the Protection of Privacy and Transborder Flows of Personal Data: Background*. Available at <http://oecdprivacy.org>.
 28. *Official Journal of the European Communities* (2001), “REGULATION (EC) No 45/2001 OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL of 18 December 2000 on the Protection of Individuals with Regard to the Processing of Personal Data by the Community Institutions and Bodies and on the Free Movement of Such Data.” Available at <http://eurlex.europa.eu/LexUriServ/LexUriServ.do?uri=OJ:L:2001:008:0001:0022:en:PDF>.
 29. United States Government (1998), *Children’s Online Privacy Protection Act of 1998 (COPPA)*. 15 U.S.C. §§ 6501–6506 (Pub.L. 105-277, 112 Stat. 2581-728, enacted October 21, 1998). Available at <http://www.ftc.gov/ogc/coppa1.htm>.
 30. Doyle, C. (2008), *CRS Report for Congress—Cybercrime: A Sketch of 18 U.S.C. 1030 and Related Federal Criminal Laws*, Updated February 25, 2008. Available at <http://fpc.state.gov/documents/organization/103707.pdf>.
 31. Microsoft Corporation (2008), *Privacy Guidelines for Developing Software Products and Services*, Version 3.1; September 2008. Available at <http://www.microsoft.com/en-us/download/details.aspx?id=16048>.
 32. Microsoft (2011), *Simplified Implementation of the Microsoft SDL*. Available at <http://www.microsoft.com/en-us/download/details.aspx?id=12379>.

设计和开发 (A4): SDL 活动与最佳实践

本章将讲解安全开发生命周期中的设计和开发阶段 (A4) 的 SDL 活动 (参见图 6-1)。这个阶段可以参照典型软件开发生命周期中的“就绪”阶段。我们将延续之前阶段的惯例,从对这个阶段的策略依赖分析开始,之后讲解安全测试用例执行的基本元素。建立合适的安全测试流程,需要已经完成创建、文档编写和测试工作,分析会持续进行,直到使软件达到所需的安全级别的调试内容被确定。接下来会讲解自动化测试工具的使用,如静态测试工具、动态测试工具、模糊测试工具,这些工具能够以较低的花费帮助我们高效和自动化地增强安全实践。静态分析在源代码编译前进行,提供可扩展的方法对代码进行安全检查,以保证代码遵循了编码安全策略。动态测试监控应用行为,确保软件功能与设计一致。模糊测试是一种故意向应用程序中引入畸形或随机数据,从而引发程序错误的测试方法,拥有高效、低成本的特点,在整个 SDL 过程中,特别是在软件版本发布前使用模糊测试寻找潜在安全问题。模糊测试是一种特殊形式的动态测试。使用最新版本的自动化测试工具以及最新的自动化安全分析方法,可以确定软件中的安全漏洞,以及需要的保护措施。在使用多种自动化工具对软件的缺陷和安全漏洞进行快速分析后,接下来应该对代码进行人工审查,以验证每个发现的问题,从而克服自动化工具和技术自身的局限性。作为这个过程的一部分,攻击表面和威胁模型的检查用来确认是否能够抵御任何设计导致的任何新的攻击向量或实现变更已经确认并缓解。最后,我们讨论在这个 SDL 阶段执行的用户隐私的验证和修复的必要性、价值和过程。

6.1 A4 策略一致性分析

这部分工作是之前章节所讲的 A3 策略依赖检查的延续。我们会在不同阶段重复地做策略分析和检查工作。策略依赖分析是最为重要的工作,你要坚持通过实际的工作检查之前的迭代是否覆盖了所有策略,而不是通过假设。在这个步骤中,你会惊奇地发现之前的阶段和迭代中会遗漏多少东西。

在这个阶段,任何存在于 SDL 域之外的策略都要被检查(或重新检查)。这些策略可能包含当在组织中开发软件或应用时来自开发组织之外的策略,这些组织维护要遵守的安全与隐私需求以及指导方针。在开发过程中策略更新以及需求增加也是经常发生的事。因此你最好能够对照策略更新列表并确认策略中已经包含了新的需求。

公司的安全和隐私策略可能指导设计者与开发者认识安全和隐私特征需要达到什么样的要求,以及实现的必要性。其他策略可能会关注当做软件产品的一部分使用的第三方以及开源软件,以及组织内外源代码以及其他知识产权的保护和控制。假如把软件安全组从核心信息安全组中分离出来的,就要特别注意这两个组关于组织支持的开发和版本发布安全的策略和指导方针的协作沟通工作。这会帮助信息安全组在公司安全策略/实践方面以及软件开发方面更好地调整策略。这也对你的公司隐私功能的协作非常重要,无论它在核心组内部还是在法律顾问组外部。如果公司发现了新的市场,隐私策略和实践应该针对那个特殊的市场进行更新。

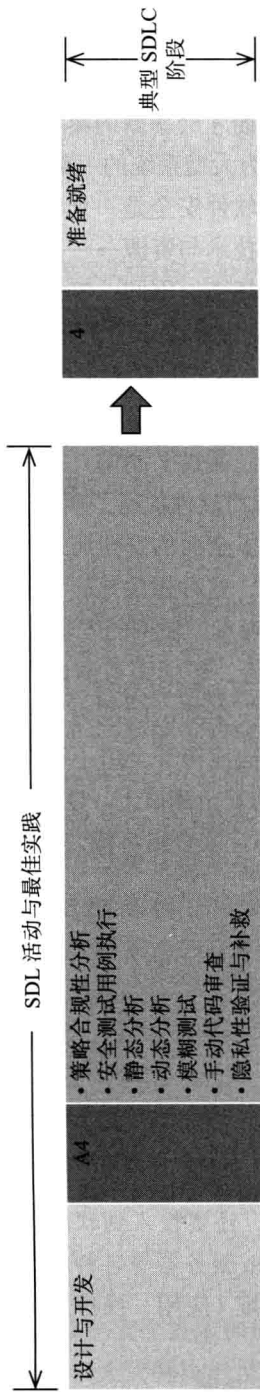


图 6-1 设计与开发 (A4): SDL 活动与最佳实践

6.2 安全测试用例执行

安全测试是一个耗时的过程，需要适当的准备，确保前后一致，并协调所有利益相关方，以及对什么是真正测试内容的深刻理解。安全测试很早就开始，并贯穿整个 SDLC 过程。安全测试方法不同于其他形式的测试，因为安全测试的目的是确认软件设计中由于不合理的设计或代码编写问题而暴露的各种安全漏洞。本书的前提是确保源代码的安全，通过这个层面的测试，软件能够防止许多只能在网络层面才会暴露的典型安全问题。安全，特别是在设计层面的安全分析，能够帮助我们在程序成为大型系统的一部分并且需要高昂的修复代价之前，发现潜在的安全问题和可能造成的影响。软件安全是一个动态的风险管理过程，需要平衡修复安全问题的花费，以对抗攻击者拥有的技术与资源——一直会有聪明的攻击者把精力投入在挖掘你软件的安全漏洞并对软件造成破坏上。因此，安全测试必须基于利用漏洞以及可能造成的相关的安全风险评估每个事件的发生概率。

在典型的 SDLC 周期中，软件需通过 QA（质量保证）测试，包括单元测试、性能测试、系统测试，以及回归测试。如果软件满足 QA 测试标准，软件就会被 QA 组赋予“通过”的结果。基本上，这就意味着软件的质量已经通过了测试并能够正常使用。但是根据作者的观点，QA 测试并不意味着测试的完成，只有对软件执行了所有安全测试并且通过所有的安全测试标准才意味着测试完成。软件只有通过了全面的安全测试才算质量合格的产品。很多公司依然把威胁安全测试当做一个额外的测试，这种做法是错误的。一旦 QA 测试完成，软件才被交付安全组进行安全测试。根据我们的观点，惯例的安全测试应该成为 QA 周期的一部分。QA 组应该把安全测试像其他测试一样对待，包括编写测试用例以及使用自动和人工方式进行安全测试。但是 QA 组往往不具备执行安全测试的能力，这意味着 QA 组依赖安全组执行所有测试。这种方式并不十分有效，因为这使得安全组不得不将大量本该用在应对高级威胁 / 个案上的测试时间，花费在了执行基本的安全测试上。QA 安全测试并不意味着取代安全组的安全测试。相反，QA 安全测试应该看作让安全组将精力集中在高级测试上的保证。下面是 QA 安全测试应该查找的一些条目：

- 配置文件中的明文口令 / 弱口令
- 堆中的默认账户（Apache、Tomcat、操作系统）
- 日志文件中的敏感信息
- 输入认证（XSS、SQL 注入）
- Web 应用的参数篡改
- 软件组使用的不安全服务（如 Telnet）
- 各种服务的安全配置（如 NFS）

安全组不仅应该把精力投入到应用上，还要投入到软件运行的栈上。这意味着从安全的观点看，还要对操作系统、相关服务、Web 服务器等各种相关对象进行配置测试。在 QA 组为产品给出“通过”的结论前，应对整个堆（应用、操作系统、Web 服务器、存储）进行基础安全项测试。

安全测试用例执行主要从两个方面进行开展。

1. 安全机制：确保安全机制的功能已经实现。
2. 安全测试：在理解和模仿攻击者的方法后，作为验证威胁模型和基于风险分析实施。

以下是典型情况下对软件产品和相关系统的三种特定的测试类型。

1. 基准测试：基准测试用来比较系统的评估状态和实际状态。

2. 预定测试：预定测试是对软件和相关系统强制性的安全性验证，无论是否是安全问题或漏洞都必须实施检测和调试。

3. 探索测试：探索性测试强调每个测试者的个性自由和对于测试质量的责任，通过学习测试相关的知识、测试设计、测试执行、测试结果解释持续地优化测试质量，在项目中采用相互支持的方式并行地执行测试工作。^[1] 测试人员积极地控制测试设计，并且在测试过程中设计新的和更好的测试方法。

成功的安全测试执行计划假设：

- 你已经完成了对软件和相关系统风险的细节分析。这个过程细节参见第 5 章。
- 测试资产已经成为风险管理计划和安全工程 / 开发测试策略的一部分。

成功的安全测试执行包含以下要素。

- 基线和基准测试已经执行，确保已经在测试周期的早期确认明显的安全问题。
- 已经验证自动化测试脚本是正确的。
- 在系统调试后重新进行过基准测试。
- 建立起未来测试的参照基准。
- 安全测试用例的执行结果已经分析。

■ 测试执行结果已经评估。你经验丰富的软件安全架构师在这项工作中扮演关键角色，需要使用他们的技能和经验与之前的测试结果进行比对，寻找和分析明显或潜在安全问题，并根据他们过去的经验来评估代码调试造成的影响。这项工作与其说是科学不如说是艺术，为了进一步评估优化调试的分析结果，应该让测试人员和开发人员在安全架构师给出初始检查和评估之后参与到这项工作中来。

■ 你已经确定是否达到安全测试用例执行验收标准。这个工作指把最新的测试结果与验收标准进行比较。如果所有结果达到或超过了安全测试执行标准，那么测试就结束了；如果没有，测试组应该继续评估测试结果。

■ 你已经确定安全测试用例执行结果是否有定论。如果结果没有达到验收标准，那么测试也许没有完结，因为测试结果不能重复而且你不能确定是什么引起了软件中的安全问题。如果结果没有完结，就需要进行探索测试。

■ 你已经确定在此刻是否需要调试。在这个阶段，任何一个已经检测出来的安全问题或更多测试需要验证是否符合额外的验收标准。要么最后的测试将再执行一次以验证结果是否可以复现，要么你将继续调整。

■ 一些测试中没有发现安全问题，因此不需要调试，但是软件必须测试，因为需要强制验证软件以及相关系统的安全性能否抵御明确已知的软件安全问题和安全漏洞。

成功的安全测试用例执行完成标准包含以下内容。

- 经过测试，软件已经达到了具体的安全需求和目标。
- 当出现超出了软件安全组控制、无法解决的情况时，如果公司股东为软件的开发负责并承担风险，当出现以下情况时，安全测试工作可以视作完成。
 - 妨碍软件安全组达到安全测试用例标准的情况在安全组控制或合同责任之外。
 - 预先约定的最终期限已经到达，并且公司股东为软件开发负责并接受未符合安全标

准的风险。在很多情况中这是可以接受的，如果下次软件升级的补丁版本中修复漏洞的责任已经被列入日程。

- 软件安全组和所有其他股东认为，尽管一些安全需求和目标没有达到，但是应用运行情况可以接受。在之前情况中，这是不能接受的，除非在下次软件升级和版本发布时修复漏洞的工作被提上日程。

6.3 SDLC/SDL 过程中的代码审查

代码审查对于发现软件开发过程中的安全缺陷非常有效。适当地执行代码审查对于代码安全起到的效果甚至比所有其他活动加起来还要多。代码审查可以让你在代码测试或安装之前就找到和修复大量安全问题。有4种分析软件应用安全性的基本技术：自动扫描、人工渗透测试、静态分析，以及人工代码审查。当然，所有这些方法都有各自的优点、缺点、擅长的方面，以及无法胜任的方面。总的来说，这4种安全代码审查方法可能是最为快速和精确的方法，它们能够发现和确定大量安全问题，同时花费较少的投入和时间。如果计划和管理得当，代码审查是性价比非常高的，特别是如果代码审查放到SDLC过程中，就可以避免接下来的开发阶段或软件产品发布后处理、定位、修复安全缺陷等环节中的较高花费。应该对有经验的、充分了解这4项代码审查基本技术的安全人员进行充分授权，以保证各种技术能够混合使用从而建立性价比最高的、管理良好的、合适的测试计划，识别软件开发过程中所有潜在和已知重要的安全问题。在这个过程中人为因素也将有助于定位自动化工具找到的安全漏洞。这种全面的方法能够找到安全问题或潜在的安全隐患，证明它们是可利用的并通过检查代码来证实。这种方法的另一种好处是，它能够帮助软件开发组获取安全开发的经验，从而防止未来出现类似的安全问题。这个过程可以在图6-2展示的4个基本步骤完成后结束，并将在下面具体描述。

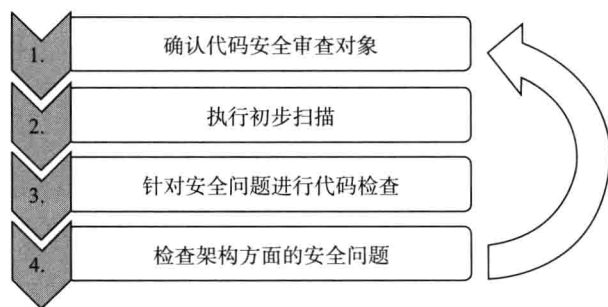


图 6-2 4 个步骤的代码审查周期

1. 确定安全代码审查对象

在这个步骤中，需要建立起代码审查的目标和限制，用来作为整个测试计划的根据。对于大多数项目来说，没有计划的基本原则很可能导致项目失败。测试计划原则特别适用于代码量巨大的复杂应用中，如 SaaS/ 云应用。集中的代码审查是有效的。这是具体的目标、时间限制、发现安全问题的知识对于代码检查如此重要的原因。减少审查的时间需求将显著增加你成功的可能。制定一个计划将会让代码审查人员每次都集中精力集中在审查代码上，这是一个有意义的改变，比尝试一次找到所有安全问题或等到项目尾声一次审查所有内容更好。

为了提高效率,你应该在审查前确定之前审查中所有已经知道的安全对象,区分已经在之前审查中提升过的安全缺陷类型,否则会把精力浪费在安全对象范围之外。

如果在其他安全工作的基础上制定代码审查计划和执行代码审查会取得更好的效果,比如之前章节提到的威胁模型。威胁模型能够确认代码的高威胁区域,使之成为需要进行仔细审查的对象,审查结果同样用来确认威胁模型中的具体假设,并且帮助理解应用的功能、技术设计、存在的安全威胁和对策。安全代码审查应该从威胁模型和设计说明开始审查,然后再审查源代码。

Chmielewski 等在《MSDN》杂志上对理想的代码审查活动流程有过描述。

- (a) 威胁模型:理解代码和数据流;确定高风险区域和输入点。
- (b) 代码审查:以适当的方式收集所有文档和程序。
- (c) 解决问题:与代码产权人合作解决发现的问题。
- (d) 吸取教训:升级工具,培训开发组,改进程序,计划未来的迭代工作。^[2]

为了使代码审查更加有效,对以下内容设定目标和限制。

- 时间:设定时间限制。这将帮助你避免在一个问题上花费太多的时间。执行多次小型的审查比执行一次很长的审查效果更好。
- 问题类型:确定寻找什么类型的问题。比如,考虑:
 - 影响保密性、完整性、可用性的主要问题。
 - 关于应用安全服务质量需求的问题
 - 关于应用服从需求的问题
 - 关于应用使用技术的问题
 - 关于应用功能暴露的问题
 - 找到的威胁模型的问题

- 查找范围外的内容:确定什么是不需要找的。解释为什么这些在你的查找范围之外。^[3]

2. 执行初步扫描

静态扫描用来发现最初的安全问题,同时能够加深你对在接下来更加完整的代码审查步骤中最有可能发现额外安全问题的位置的理解。这是通常通过自动扫描、人工扫描或者两种扫描完成,如何开展工作依赖于代码审查计划中的审查对象和时间限制。自动扫描在大型应用中具有快速确定“好摘的果子”的优点。自动扫描的结果可以用来建立需要优先审查的潜在安全漏洞和安全机制的列表。

自动扫描可以作为人工审查的补充,对于代码量很大、人工花费时间或成本过高的软件,可以在自动扫描确认代码区域后人工集中处理,自动扫描还可以发现人工审查可能会漏掉的安全问题。自动扫描工具特别擅长对单行代码、多行组成的单个函数进行测试,这些内容在人工审查中很容易遗漏。尽管自动扫描能够辅助人工审查,它依旧不能替代人工审查,因为它无法处理上下文问题、无法在多个函数中查找安全问题(如在 SaaS/云应用中)。自动化工具的另一个问题是它会有大量的漏报和误报。对于误报,这类结果可以使你在审查代码判断是否误报时强化你对代码的理解,包括控制流和数据流。使用自动化工具的另一风险是,如果没有检测出安全问题就会认为软件是安全的。永远不要假设一个庞大和复杂的软件产品没有安全漏洞。所有可能的步骤只是尽量减少代码错误的数量并降低由此导致的实际影响,但是有些问题总是会遗漏的。自动化安全测试工具可以发现大量代码错误,但是会遗漏一些安

全漏洞，这些安全漏洞隐藏在大量的误报中无法被检测出来。人工源代码分析无法替代这些工具，但是当人工源代码分析与这些工具相结合时会最大化地检测出安全漏洞。就像这本书里提到的，它需要一个整体的安全方法，包括人工因素，用来保证误报确实是误报，并且代码确实不包含无论使用什么自动化检测工具都无法发现的安全漏洞。两种方法结合起来能够高效率地、高性价比地查出更多的安全漏洞。由于这些原因，自动审查结合人工审查是最好的方法。

为了捕获更简单安全问题，在人工审查之前应先使用自动化工具对代码实施检测。为了防止发现已经知道的安全问题，所有之前确认的安全漏洞都应被修复。接下来实施人工审查，用来更好地理解代码并识别模式。扫描结果确认值得代码审查人员进一步分析的区域，代码中需要分析的安全问题在下面的第3步中。但是这个工作应该占所有代码审查工作时间的很小比例，应该集中精力解决以下问题。

- 输入数据验证：应用程序是否有输入验证机制？验证机制是否在客户端、服务端都能起到效果？是否有验证核心机制，验证规范是否已经覆盖了代码库？
- 负责验证和授权用户的代码：应用程序是否验证用户？允许什么角色？他们之间如何互动？是否有订制的验证和授权代码？
- 错误处理代码：是否有持续的错误处理机制？应用程序能否获取和抛出结构异常？是否有错误处理特别多或特别少的代码区域？
- 复杂代码：是否有代码特别复杂的区域？
- 加密：应用程序是否使用加密？
- 互操作性：应用程序是否使用互操作性调用本地代码？^[4]

3. 针对安全问题进行代码检查

步骤2的结果可以用来帮助分析人员集中分析步骤3。

4. 检查架构方面的安全问题

这是软件安全架构师或经验丰富的软件安全专家施展才华的地方。如果你不具备专业技能，可能会用到第三方人员。专家应用他们对业务逻辑的知识、使用或滥用用例、之前工作的经验来确认安全漏洞从而降低误报和漏报的可能性。静态分析工具对查找应用程序的缺陷和业务逻辑缺陷的安全隐患无能为力，需要对应用程序理解的人员进行确认。经验丰富的安全专家使用SDLC过程可以平衡开发者可能会忽略某些代码错误的思维习惯，尽管他们编写了具体的代码并且最为了解它们。经验丰富的专家能够帮助理解代码使用了哪些技术，不仅包括软件产品使用的具体技术，还包括操作系统、第三方的使用工具以及开发工具。从安全的观点看，这些安全专家可以确认一款产品和其他系统、应用软件或服务器之间的关系。在安全的背景下，他们可以确认一款软件产品依赖什么组件，什么软件依赖这款产品，以及它如何利用这些关系用来确认一款产品影响剩下的系统，以及它可能如何被影响。人为错误能够引起大部分安全问题；使用目前有缺点的自动化测试工具，人力因素应该帮助解决问题。例如，一个小的编程错误能够导致严重的安全漏洞，从而危害整个系统或网络的安全，由于开发周期过程中的一系列错误，引入编程错误并且没有在测试阶段发现，同时可用的防御机制无法阻止攻击者成功攻击系统。这是另一个说明在该过程中需要人力因素的例子，这种情况超出了自动化安全测试工具的能力范围。

一些产品的基础设计也许包含缺陷，这些应该被记录，尽管这些会影响产品的可靠性，

但这些代码错误并不是安全漏洞。记住，安全检查的最终目的是寻找代码中能够被攻击者利用进而绕过安全边界的安全漏洞。

6.4 安全分析工具

代码安全审查的最终目的是提升产品整体的安全性，使开发组提升安全开发的能力从而减少代码中犯的错误。本节讨论各种测试方法在整个过程中的功能和角色的细节，包括静态分析、动态分析、模糊测试以及人工代码检查。但是在开始前，我们需要认识到每种方法都有各自的优点和局限性。

代码静态分析的优点

1. 使用现有的命令可以使软件执行
 - 不需要猜测和解释行为。
 - 能够产生所有软件可能产生的行为。
2. 能够找到代码缺陷的精确位置。
3. 能够被培训过的完全理解代码的软件保障开发人员使用。
4. 允许快速修复发现的问题。
5. 自动化工具运行速度很快。
6. 自动化工具能够扫描整个代码库。
7. 自动化工具能够提供修改建议，减少研究问题的时间。
8. 能够在开发周期早期就发现问题，减少修复花费。

静态代码分析的局限性

1. 需要使用代码或者至少是二进制代码，静态代码分析通常需要使用足够的软件工件来执行构建。
2. 静态分析通常需要熟练地执行软件构建。
3. 不会发现与操作配置环境相关的问题。
4. 如果人工执行会花费大量时间。
5. 自动化工具不支持所有程序语言。
6. 自动化工具存在误报和漏报。
7. 没有足够多受到培训的人员从事彻底执行静态代码分析的工作。
8. 自动化工具会给人所有问题都已经解决的安全错觉。
9. 自动化工具只能达到扫描规则的水平。
10. 无法找到运行时环境引入的安全缺陷。^[7, 8]

动态代码分析的优点

1. 限制发现问题的范围。
 - 应用程序必须留下寻找测试区域的轨迹。
 - 这会导致遗漏区域。
 - 只能测试已经发现的。
2. 不使用实际命令执行。
 - 工具使用应用程序。
 - 对请求与回应进行模式匹配。

3. 仅仅需要一个运行的系统即可执行测试。
4. 不需要使用源代码或二进制代码。
5. 不需要理解如何编写软件或构建程序。
 - 大多数工具只要人工运行起来就可以不管了。
6. 测试一个具体的操作部署。
 - 能够发现静态分析工具会遗漏的基础结构、配置和补丁错误。
7. 在运行时环境下确认软件的安全漏洞。
8. 自动化工具能够灵活地确定扫描对象。
9. 对应用程序的分析可以不使用程序代码。
10. 能够确认在静态分析中漏报的漏洞。
11. 对静态代码分析结果进行确认。
12. 能够对任何应用程序进行测试。^[9, 10]

动态代码分析的局限性

1. 自动化工具会给人所有问题都已经解决的安全错觉。
2. 自动化工具会有误报和漏报。
3. 自动化工具只能达到扫描规则的水平。
4. 就像静态分析，没有足够多受到培训的人员从事彻底执行动态代码分析的工作。
5. 安全漏洞回溯到代码中精确位置更为困难，需要花费更多时间来修复问题。

如果你无法接触源代码或二进制代码，不是软件开发人员，不理解软件构建，或者对操作环境进行简单测试，你可能会选择动态测试工具；否则，你可能会选择静态分析工具。理想情况下，两种测试工具都应该使用。

模糊测试的优点

1. 模糊测试最大的好处是测试设计极为简单，你不需要预先知道系统行为。
2. 这种系统化 / 随机方法往往可以发现人工无法发现的 bug。另外，当测试系统完全关闭（如 SIP 手机）时，模糊测试是唯一的检查方法。
3. 使用模糊测试找到的 bug 往往是很严重的，可以被真实的攻击者加以利用。模糊测试被越来越广泛地被人们知道，因为这种技术和工具目前被攻击者用来发现软件的安全隐患。这是模糊测试具有的主要优势，相比二进制代码或源代码审计，以及模糊测试最为接近的测试方法——故障注入——依赖于人工故障条件因此发现的问题很难甚至无法被利用。

模糊测试的局限性

1. 模糊测试工倾向于查找简单的 bug；另外，适用于特定协议的模糊测试越多，能否发现奇怪错误的数量就越少。这就是彻底 / 随机方法依旧很流行的原因。
2. 另一个问题是，当你做黑盒测试的时候，你往往攻击的是一个关闭的系统，这就增加了评估发现的安全问题的危险 / 影响（不是指修复 bug 的可能性）。
3. 用模糊测试查找程序错误的主要问题是，找的总是非常简单的错误。问题是指数增长的，每个模糊测试工具都有在时间表中找到人们感兴趣的问题的捷径。较早的模糊测试工具可能有比较低的代码覆盖率；比如，如果输入包含一个没有经过适当升级后用来匹配其他随机信息的校验和，只验证通过校验和的代码。代码覆盖工具经常用来评估模糊测试工具的质量，但是这只是一个参考。每个模糊测试工具都能找到不同类型的 bug。^[13, 14]

人工源代码审查的优点

- 1. 不需要技术支持。
- 2. 能在很多情况下使用。
- 3. 灵活。
- 4. 促进团队协作。
- 5. 在 SDLC 的早期。

人工源代码审查的局限性

- 1. 花费大量时间。
- 2. 并不是一直有合适的条件。
- 3. 人员需要很好的技能才能够有效。^[15]

6.4.1 静态分析

静态程序分析是在计算机软件实际上不执行的情况下进行的测试。静态分析主要用在某一版本程序源代码的分析上，它也会用在对象代码上。相比较，动态分析实际上是在软件程序执行后进行测试的。静态分析是使用自动化工具进行测试的，不要与人工分析或软件安全架构检查混淆，后者一般包括人工代码审查，需要对程序有一定理解。当静态分析使用得当时，它相比人工静态分析的显著优势就会显现出来，它可以使用比开发者更多的安全知识更频繁地进行测试。而当确实需要时再使用专业的安全架构师或工程师。

静态分析（见图 6-3）也称为静态应用程序安全分析（SATA）。它在开发和质量保障（QA）阶段对安全漏洞进行确认。SATA 提供代码行级别的检测，确保开发组可以迅速修复安全漏洞。

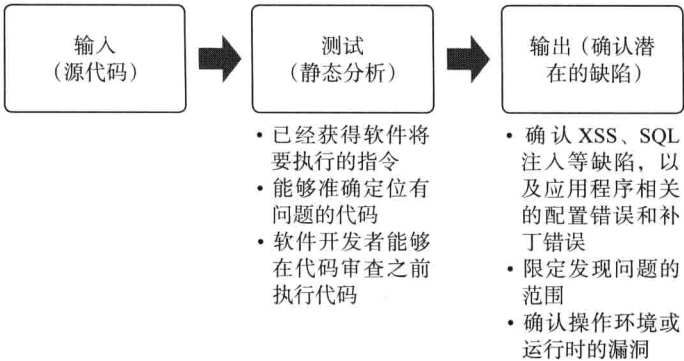


图 6-3 静态分析流程图

成功的另一个关键因素是使用静态分析工具和选择适合于测试环境的测试工具厂商。能够对软件任何部分进行自动化测试的技术都是受欢迎的，但是由于没有选择正确的人和 / 或正确的步骤对测试工具进行选择，软件在一些机构中就会变成“闲置的”。并不是所有工具都是一样的：有的工具比较善于以特定语言编写的程序，而一些工具擅长前端 GRC（控制、风险管理、一致性）以及度量分析功能。一些情况下，你需要使用三种不同的测试工具以保证测试的有效性。一些受欢迎的 SAST 厂商产品有 Coverity^[16]、HP Fortify Static Code Analyzer^[17]、IBM Security AppScan Source^[18]、klocwork^[19]、Parasoft^[20]，以及 Veracode。^[21]

使用静态分析工具的一个挑战是，当分析一个与闭源组件或外部系统进行交互的应用时，

误报往往会写在报告中，由于没有源代码，不可能对外部系统中的数据流进行跟踪并确保数据的整体性和安全性。静态代码分析工具有时也会有漏报，安全漏洞存在但工具无法找出它们。这可能发生在以下情况下，如果在一个新的外部组件中安全漏洞，或如果分析工具没有运行时环境以及它是否安全配置的知识。静态检测工具经常会出现误报，即工具报告存在但实际上并不存在的安全漏洞。这种情况将常出现，因为工具无法确定通过应用程序的输入、输出时数据的整体性和安全性。^[22]

Michael Howard，在他的 2006 年 IEEE 的安全与隐私大会上发表了名为“执行代码安全审查流程”的文章，建议用如下辅助办法确定代码审查优先级。这些办法可以用来作为确定静态测试、动态测试、模糊测试和人工代码审查优先级的指导。

- **旧代码：**旧代码存在比新代码更多的安全漏洞，因为新代码对安全问题了解更多。所有“遗留”代码都应该进一步审查。
- **默认运行的代码：**攻击者从运行默认安装的代码进行攻击。这部分代码应该比其他代码更早、更深入地进行审查。默认运行的代码增加了应用程序的攻击表面。
- **在高级环境下运行的代码：**在高级标识（如 *nix 中的 root 用户）下运行的代码，也需要更早和更深入地审查，因为代码标识是另一个攻击表面的部分。
- **匿名用户涉及的代码：**涉及匿名用户的代码应该更深入地审查，因为只有合法用户和管理员才能够进入系统。
- **全局监听网络接口通信的代码：**默认监听网络的代码，特别是不受控的网络，例如互联网，是潜在的风险，必须深入检查安全隐患。
- **使用 C/C++/ 汇编语言编写的程序：**由于这些语言编写的程序能够直接访问内存，存在操作缓存的风险，可能经常用于恶意代码执行的情况，如导致缓冲区溢出。用这些语言编写的代码应该深入地分析是否有缓冲区溢出漏洞。
- **有安全隐患历史的代码：**过去存在很多安全漏洞的代码应该被怀疑，除非能够证明这些安全漏洞已经被有效地移除。
- **处理敏感信息的代码：**处理敏感信息的代码应该被分析，以确认这些代码不存在将这些数据暴露给不可信用户的弱点。
- **复杂代码：**复杂代码有 bug 的可能性很高，因为这些代码更加难懂，因此存在安全漏洞的可能性更高。
- **频繁更改的代码：**代码频繁更换经常导致新 bug 的引入。这些 bug 不一定是安全漏洞，但是与不怎么更新的稳定代码相比，这些不稳定的代码更可能出现漏洞。

在 Michael Howard 的 2004 微软大会上的名为“最小化暴露给不可信用户的代码以降低安全风险”的文章中，他建议一种概念性的三阶段分析过程以优化静态工具的使用。

阶段 1：运行所有可用的代码分析工具

- 多种工具应该互相弥补各自的局限性，尽量减少漏报和误报。
- 分析过程中应该注意每个警告或错误。
- 如果多个工具都对一段代码有警告，说明这段代码需要进一步的安全检查（比如人工分析）。代码应该尽早评估，最好对每个版本都进行评估，并且在每个里程碑都进行重新评估。

阶段 2：寻找共同的安全漏洞模式

- 分析人员应确保代码审查涵盖了最多的安全漏洞和缺陷，例如整数运算问题、缓冲区

溢出、SQL 注入、跨站注入 (XSS)。

- 这些公共安全漏洞和缺陷的信息来自公共漏洞披露 (CVE) 和公共缺陷枚举 (CWE) 数据库中, 由 MITRE 公司进行维护, 链接是: <http://cve.mitre.org/cve/> 和 <http://cwe.mitre.org>。
- MITRE 是与 SANS 合作的机构, 也维护一个“排名最高的 25 个最危险的程序错误”列表 (<http://cwe.mitre.org/top25/index.html>), 这些程序错误会导致严重的安全漏洞。
- 静态代码分析工具和人工技术至少应该定位“排名最高的 25 个最危险的程序错误”中的错误。

阶段 3: 深入分析风险代码

- 分析人员应该使用人工分析 (如代码检测) 对基于攻击表面风险代码或之前讨论过的启发式问题已经确认的任何风险代码, 进行更为彻底的评估。
- 代码审查应该在每个模块的入口点进行审查, 并跟踪通过系统的数据流, 评估数据, 它是如何使用的, 以及是否危害安全对象。

以下是一个通过静态分析找到的安全问题的例子。注入漏洞在 OWASP 2013 年排名最高的 10 个安全问题列表中。这类漏洞发生的原因是, 不可信数据作为结构化的命令未经确认就直接用来查询。注入漏洞有不同的类别, 如 SQL、OS 和 LDAP。如果用户输入直接用来构建一个 SQL 查询, SQL 注入攻击就会成功。

用户检查他账户的细节。应用程序需要他的用户 id 或标识符从后端数据库查询账户信息。应用程序可以通过一个 URL 链接参数绕过身份鉴别过程, 例如:

```
http://example.com/application/reviewaccount?account_id='1007'
```

在这个例子中, 应用程序获得用户 `account_id` 为 '1007', 并使用这个 id 获取数据库的信息。后端查询是这样的:

```
String insecureQuery = "SELECT * FROM accounts WHERE  
accountID=" + request.getParameter("account_id") + "'";
```

如果恶意用户将参数的值改为 'or '1'='1', 接下来不安全的字符串查询会获得这个值:

```
SELECT * FROM accounts WHERE accountID=' ' or '1'='1';
```

'1'='1' 会永远为真, 因此这个查询结果会产生所有账户信息。这显然不是开发者的意图, 但是通过可信用户输入建立查询, 他允许恶意用户执行任意数据库命令。

静态分析工具对代码的分析能够确认用户输入构成的查询能够导致 SQL 注入攻击。

6.4.2 动态分析

动态程序分析是通过在真实或虚拟处理器上运行程序进行计算机软件分析的方法。动态测试的目标是在程序运行时找到程序的安全问题, 而不是在软件离线时重复检查代码。通过在设计的所有场景下调试程序, 动态分析无需人工构建可能会产生错误的环境。动态分析对于确认静态分析结果中误报的漏洞有很大的优势。

动态分析 (见图 6-4) 也称为动态应用软件安全检测 (DAST)。它用来确认应用软件产品中的安全漏洞。DAST 工具用来快速评估系统的整体安全性, 在 SDL 和 SDLC 中都会

用到。动态分析与静态分析有相同的优点和注意事项。一些受欢迎的 DAST 厂商的产品包括 HP Webinspect^[26] 和 QAINspect^[27], IBM 的 Securit AppScan Enterprise^[28], Veracode^[29] 和 Whitehat Sentinel Source^[30]。

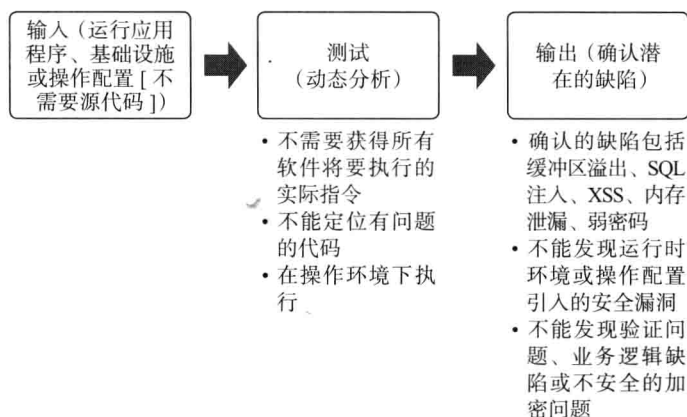


图 6-4 动态分析流程图

接下来对于贯穿 SDLC 中如何使用动态分析的说明来自 Peng 和 Wallace (1993) NIST 特殊出版物 500-209, 《软件错误分析》。^[31]

- 通常在设计阶段使用动态分析技术包括粒度和时间分析、原型分析，以及模拟。粒度和时间分析对于实时程序的分析很有用，包括响应时间需求、内存限制、执行空间需求。这种类型的分析尤其适用于确定硬件与软件的协作是否符合设计架构；在系统测试中发现由系统基础设计引起的性能问题需要花费大量的成本。自动模拟适合大型的设计。原型分析能够用来帮助检查整体的设计架构或一组特定的函数。对于大型的、复杂的系统，原型分析可以避免不适合的设计导致耗费大量系统资源的情况。^[32]
- 动态分析技术帮助确定代码的功能和计算的正确性。回归分析用来在重要代码编写完成后重新评估需求和设计条目。这种分析确保对原始系统需求的认识。粒度和时间分析在代码增量开发过程中执行，将分析结果与预测值进行比对。^[33]
- 在测试阶段的动态分析涉及不同类型的测试方法和测试策略。传统上有 4 种测试类型：单元测试、集成测试、系统测试、验收测试。单元测试是对软件单元、模块或子程序的结构性或功能进行的测试。结构性测试检测软件单元的逻辑，用来支持测试覆盖的需求——程序有多大程度被执行。功能性测试评估软件实现了多少软件需求。功能性测试中，测试人员往往没有任何程序设计信息，因为测试用例是基于软件需求的。^[34]
- SDLC 最后阶段的最为常用的动态分析技术是回归分析和测试、模拟，以及测试认证。当这个阶段中产品发生了改变时，回归测试用来证明程序没有违反影响程序其他区域的基本需求和设计假设。模拟用来测试操作员步骤和隔离配置错误。测试认证，特别是在关键软件系统中，用来证明需要的测试已经执行完毕，交付的软件产品与认证和验证的软件产品是一致的。^[35]

静态分析工具通过需要分析源代码查找问题。动态分析工具不需要源代码就可以确定问题。在讨论静态分析时，我们介绍了一个 SQL 注入攻击例子。在那个例子中，工具能够确认那个 account_id 能够以 URL 参数的形式通过验证、尝试篡改参数的值并评估应用程序的响应。

6.4.3 模糊测试

模糊测试（见图 6-5）是一种自动化或半自动化的黑盒软件测试技术，它为计算机软件输入提供非法的、非预期的数据或随机数据。换句话说，模糊测试通过自动化方式注入不正确的、半不正确的数据寻找软件 bug 或安全缺陷。软件程序的输入用于监控是否有异常返回，如程序崩溃、内置的代码断言错误，以及潜在的内存泄漏。模糊测试已经成为测试软件或计算机系统安全问题的关键元素。模糊测试比起其他测试方法的一个明显优点是测试设计极为简单并且对系统行为没有偏见。

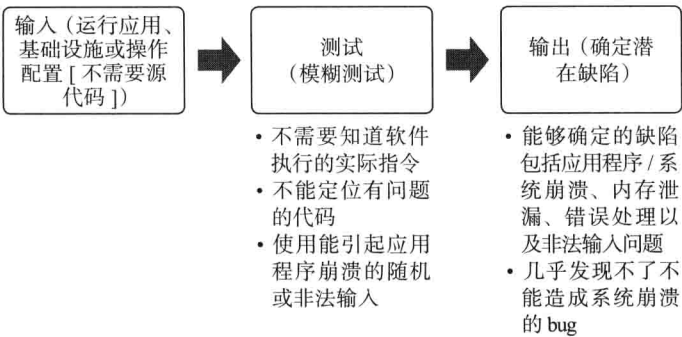


图 6-5 模糊测试流程图

模糊测试是软件安全的关键因素，必须嵌入在 SDL 中。很多厂商选择了这个领域，并且开发了自己的工具。有两个流行的模糊测试工具，一个是 Codenomicon^[36]，它是市场上最为成熟的商用模糊测试工具之一；另一个是 Peach Fuzzing Tool^[37]，它是最为流行的开源模糊测试工具之一。模糊测试用在安全测试和质量保障测试中。目前认为在许多软件开发程序中模糊测试是关键元素和主要缺陷，以至于它现在成为美国防御部（DoD）信息保障认证和认可流程（DIACAP）需求。

模糊测试是一种对系统开放接口输入非预期数据并对系统行为进行监控的攻击模拟形式。如果系统出错，例如崩溃或内置程序断言错误，就说明软件有缺陷。与静态分析工具不同，尽管模糊测试工具找到的所有问题都是危险的和可利用的，但是模糊测试只能找到通过开放接口进入系统的 bug。模糊测试工具必须能和测试软件进行互操作，这样模糊测试工具才能进入更深的协议层并通过测试多个层面更彻底地测试系统。

尽管静态分析对代码有完整的覆盖率，并且能够提升整体软件质量水平，但它不能简单地提供解决复杂问题的测试结果，并且就像之前讨论的，静态测试会导致大量误报，这两个问题都需要借助人力进一步分析并消耗宝贵和有限的资源。模糊测试没有误报，因为发现的每个缺陷都是一个攻击模拟的结果。

静态分析是对没有执行的代码进行的，并且只能在离线条件下执行。作为对比，模糊测试必须对可执行的代码进行，能够对运行的软件进行，因此能够找到静态代码中不易找出的安全漏洞。模糊测试的目标是攻击者也会找到的，因此模糊测试是可靠性很好的测试方法，通过将所有精力只集中在最危险的并且易受攻击的接口上优化程序。由于模糊测试具有健壮性测试的能力，在产品发布前 SDLC 的验证阶段会通常使用模糊测试。就像静态分析和动态分析，可以在软件组件刚完成或版本发布之后进行模糊测试，而不一定只在 SDLC 过程的确

定时刻。当然，这种特点能够使安全漏洞在 SDLC 的早期就被发现和修复，显著减少发现和修复漏洞花费的时间。

标准的模糊测试技术受限于其使用随机突变的方式生成测试用例，这种测试技术只能找到软件的一部分安全漏洞。但是，这项工作依然是有意义的，因为这些漏洞与攻击者发现的漏洞相同。请记住，攻击者也使用模糊测试工具，如果探测者或攻击者使用模糊测试工具在已经发布给客户的版本中发现安全缺陷，就说明这个软件安全性很差的程序存在很严重的问题。更精细的模糊测试技术用来提升和优化代码覆盖率，方法是使用与目标协议区域匹配的特定协议对最易被利用的安全漏洞进行检测，并且可以在不降低代码覆盖率的前提下减少测试用例的数量。静态代码分析用来保证代码遵守安全代码策略，而协议模糊测试工具用来揭示攻击者角度下可能存在的威胁和风险。

模糊测试的另一个优点是它能够用在黑盒测试、灰盒测试、白盒测试中，不需要使用源代码。就像本章讨论过的动态分析工具一样，这种特征使模糊测试成为不需要外包开发的第三方软件的测试中优秀的测试方法。然而，模糊测试的一个缺点是它的侵入性，很容易导致系统崩溃，在隔离的环境（比如测试实验室或虚拟环境）下这需要进行初步测试。

模糊测试分为两大类，“聪明型”和“笨拙型”。

- 在聪明型（增长型）模糊测试中，模糊测试工具以一种逻辑方式对程序进行输入，一般方式是等待系统响应和可能出现的堆改变。这种方法需要对目标系统和专用工具进行深入了解，但是比起“笨拙型”模糊测试会有更少的系统崩溃分析和重复结果。
- 在“笨拙型”（突变型）模糊测试中，模糊测试工具会系统地将数据推送给程序而不等待合适的响应。这种方法不需要对目标系统和已有工具的使用方法进行了解。但是，需要分析结果，并且比“聪明型”模糊测试有更多的重复结果。

为了实施模糊测试，每个文件或字段应循以下步骤向应用程序输入：

1. 向输入文件输入随机数据或空格。
2. 使用那个文件对应用程序进行测试。
3. 评估结果。破坏了什么？什么正常运行？什么是期望发生的？
4. 把每个测试用例和报告发现的问题报告给项目管理人员。^[42]

6.4.4 人工代码审查

人工安全代码审查是通常对软件产品进行逐行检查以发现安全漏洞的方法。这可能会包括对多层和多组件的企业软件产品的程序源代码进行彻底审查。在使用多种能够快速分析代码缺陷和安全漏洞的自动化工具后，进行人工代码审查。人工审查代码以确认每个发现的问题，从而克服自动化工具和技术的局限。尽管使用不同的方法能够发现代码错误，但是人工代码审查依然比最精密的工具有更高的精确性和质量。但是另一方面，人工代码审查也是最为耗费资源的。

人工代码审查由人工驱动，尽管人工发挥最大价值的地方是对软件架构设计的检查，即对人、策略、过程等进行软件整体安全性检查。假如资源有限，人工代码审查最好只在应用程序最为关键的部分执行。这些审查也包括对文档、安全编码策略、安全需求和安全结构设计的审查。这些工作也用来指导人工审查，其中软件安全架构师能够为开发组的其他人提供测试艺术指导，帮助他们理解安全过程、安全编码策略意识，以及设计和实现安全应用程序

的技能。就算在熟练的和对安全理解很好的开发团队，软件安全架构师仍应该采用信任-验证模型。这个过程被架构师与利益相关者、合适的开发团队成员一起对文档的分析以及架构师询问设计者与系统拥有者的输入所增强。

应该注意的是，如果使用了好的软件工程过程，代码审查团队能够减少很多需要关注的方面。在大多数情况下，静态分析、动态分析、模糊测试在发现实现层面 bug 的效率比代码审查更高，但是如果连人工审查都只能找到较少的安全漏洞，这只能说明这些漏洞只能通过人工审查才能发现。一旦一个类型的安全漏洞通过人工审查代码的方法找到，这些安全漏洞应该合并到自动化代码审查工具中。就像之前提到的，有效和高效地提高软件安全需要整体的方法，不仅包括人工代码审查，还包括强制软件安全培训、安全设计检查、威胁模型、模糊测试、静态和动态分析、高风险实践的确认、软件生命周期各种阶段可度量的标准 and 需求，包括服务和支持。

以下步骤是人工软件安全审查的典型步骤。

- 威胁模型用来确定风险，并告诉开发团队应该优先查看的代码，通过审查可以帮助开发团队理解关于软件功能存在的安全威胁。
- 以上描述的各种自动化工具用来评估代码的语义和语言安全漏洞，优化查找最高风险的过程，投入最大的工作量修复或减轻安全漏洞。
- 人工执行逐行的软件代码审查用来寻找逻辑错误、不安全的加密、不安全的系统配置和其他平台特有的问题。

使用问题驱动方法能够让检查保持活跃状态。标准问题列表能够帮助你集中精力在并非你的软件架构独有的普通安全漏洞上。这个方法用来与控制流和数据流分析等技术相结合以优化跟踪代码路径的能力，这是最有可能发现安全问题的方法。应该至少在最常见的编码漏洞层面解决问题。当你使用控制流和数据流分析时应该问自己以下这些问题。注意，发现一些安全漏洞可能需要控制流和数据流上下文的知识，而另一些可能不需要上下文知识并且使用简单的模式匹配就可以找到。当执行代码安全审查的时候，下面的一些技术可能会结合在一起：

- **控制流分析：**控制流分析是单步调试代码逻辑条件的机制。过程如下。
 1. 检查函数并确定每个分支条件。这可能包含循环、switch 语句、“if”语句和“try/catch”模块。
 2. 理解每个模块执行的条件。
 3. 查看下一个函数并重复。
- **数据流分析：**数据流分析是从输入点到输出点跟踪数据的机制。因为应用程序中可能会有很多数据流，所以使用步骤 2 中的代码审查对象和标志区域使你集中精力工作。过程如下。
 1. 对于每个输入点，确定你对输入源的信任程度。当输入源不可信时，将其归为不可信一类。
 2. 跟踪每个可能的输出点的数据流。注意任何对数据认证的尝试。
 3. 查看下一个输入并继续。^[43]

当执行数据流分析时，审查输入列表和输出列表，接下来匹配你需要审查的代码。你必须特别注意通过可信边界的代码区域以及改变代码信任等级的代码区域的优先级，就像你在威胁

模型过程中做的一样。软件可以调用的一组常见例程一接触任何不可信的数据就应该可用，它能够给你的软件产品一个核心认证区域，用来更新新发现的信息。当执行数据流分析时，尤其注意有很多输出点的数据解析区域，以保证数据跟踪回它的源，基于最弱链接分配信任。

还有其他需要考虑的问题列表。一部分整理在一组由软件产品或开发解决方案最常见的软件安全漏洞导致的实现错误关键区域，也称为热点。这些典型问题被软件安全架构师整理在之前提到的 top10 ~ 20 CVE 或 OWASP 的“TOP 10”列表中。

对安全架构独特安全问题的审查也作为人工安全审查过程的一部分执行。这个步骤尤其重要，特别是在软件产品使用定制的安全机制或特征来降低已知安全威胁。在这个步骤中，代码审查对象列表也用来检查那些还没审查的内容。这部分工作也使用问题驱动的方法，作为最终的代码审查步骤来确定软件架构中独特的安全特征和需求，见下面的列表。

- 你的软件架构中包含定制的安全实现方式吗？定制的安全实现方式是发现安全问题的好地方，有如下原因。
 - 它已经识别出安全问题，这是把定制安全代码写在第一个位置的原因。
 - 与其他产品区域不同，功能性问题很可能导致安全漏洞。
- 已知的安全威胁都降低了吗？降低已知威胁的代码需要仔细审查，因为一旦它们出现问题就会导致规避安全机制。
- 应用程序中是否包含独特的角色？使用角色假设有一些用户的权限低于另一些用户的权限。确保允许一个角色改变另一个角色权限的代码没有问题。^[44]

我们重申这并不是不同类型安全测试的建议。为了使产品安全，需要使用所有类型的安全测试——静态分析、动态分析、人工代码审查、渗透测试和模糊测试。经常会出现权衡开发周期和时间限制或最终期限后，而跳过测试阶段急于将产品投放市场的情况。这种做法节省了时间，使产品几个星期 / 月就可以发布。但是，从 ROI 的观点来看，这是要付出高昂代价的。产品发布之后发现安全问题会对客户造成很大损害，从而影响公司的声誉。

6.5 成功的关键因素

SDL 第 4 阶段的成功依赖于遵从策略，执行安全测试用例，完成不同类型的安全测试，以及验证隐私需求。表 6-1 列出了这个阶段成功的关键成功因素。

表 6-1 关键成功因素

关键成功因素	描 述
1. 安全测试用例执行	覆盖所有相关的测试用例
2. 安全测试	完成所有类型的安全测试，修复找到的问题
3. 隐私验证和修复	隐私相关控制的有效性，修复找到的问题
4. 策略合规性检查	按照阶段 4 更新策略

成功因素 1：安全测试用例执行

6.2 节提到了安全测试执行计划成功的标准。

成功因素 2：安全测试

完成所有类型的安全测试——人工代码审查、静态分析、动态分析、渗透测试和模糊测试——是很关键的。应该对每个测试类型中发现的问题进行风险评估并排出优先级。任何中

风险或高风险的安全缺陷应该在产品发布或部署前修复。同时也不应忽视低风险缺陷，而应该尽快创建修复计划。

成功因素 3：隐私验证和修复

验证隐私问题应该成为安全测试计划和安全测试的一部分。但是，使用独立的工作流评估产品隐私控制的有效性也是很好的办法。应该在产品发布和部署前优先修复任何已确认的问题。

成功因素 4：策略合规性检查（更新）

如果确认了任何额外的策略，或之前确认的策略已经根据阶段 3 进行了更新，应该对更新内容进行检查并根据计划更新产品。

6.6 可交付成果

表 6-2 列出了 SDL 这个阶段的可交付成果。

表 6-2 A4 阶段可交付成果

可交付成果	目 标
安全测试执行报告	检查安全测试用例执行的进程
更新策略的合规性分析	分析公司策略的遵守程度
隐私遵守报告	验证执行的隐私评估是否符合建议
安全测试报告	不同类型的安全测试发现的结果
修复报告	产品的安全状态

安全测试执行报告

执行报告应该符合安全测试执行和测试频率的状况。报告应该符合验证问题修复的回归测试数量。

更新策略的合规性分析

策略遵守产品分析（见第 4 ~ 5 章）应该基于在 SDL 的这个阶段任何新出现的需求或策略进行更新。

隐私遵守报告

隐私遵守报告应该支持早期阶段的隐私需求。任何未解决的需求应当尽快解决。也应该谨慎地对任何规范 / 条例的改变确定新的需求。

安全测试报告

每类安全测试应该写入一个问题摘要中：人工代码审查、静态分析、动态分析、渗透测试、模糊测试。报告应提供问题的类型和数量以及任何能从结果中溯源的内容。例如，如果在某个应用程序的组件中找到相对其他组件少很多的 XSS 问题，可能是因为之前的开发者受过良好的训练或者使用的框架更加有效。应该在下个版本周期的前一个 SDL 阶段反馈回来。

修复报告

应该为这个阶段的定期更新准备好修复报告 / 仪表盘。这个报告的目的是从技术层面展示安全的特征和产品的风险。

6.7 度量标准

应该在 SDL 的这个阶段收集以下度量标准（之前讨论过的一些应该交叉度量）。

- 遵守公司策略的比例（更新）
 - 遵守阶段 3 和阶段 4 的比例对比
- 通过静态测试工具实际测试的代码行数
- 通过静态分析工具找到的安全缺陷数量
- 通过静态分析工具找到的高风险安全缺陷的数量
- 缺陷密度（每一千行代码的安全问题数量）
- 通过静态分析、动态分析、人工代码审查、渗透测试、模糊测试找到的安全问题的类型和数量
 - 使用不同类型的交叉测试找出的安全问题
 - 对比不同测试类型找到的安全问题的严重程度
 - 把找到的安全问题与之前确定的威胁 / 风险进行映射
- 对发现的安全问题进行修复的数量
 - 发现问题的严重程度
 - 花费在修复发现问题上的小时数（大约）
- 发现问题中未解决问题的数量、类型和严重程度
- 遵守测试安全计划的比例
- 安全测试用例执行的数量
 - 执行安全测试用例发现的问题数量
 - 执行回归测试的数量

6.8 本章小结

在对设计和开发阶段（A4）的讨论中，我们讲解了成功测试用例的执行流程，使用自动化工具和人工审查两种方法审查代码的合适流程，以及在 SDL 的这个阶段中隐私验证和修复的实现流程。本章最重要的流程提供了有效和高效地测试、调整和修复所有已知和发现的漏洞，以及确保遵守安全代码策略、在产品化（A5）阶段之前提供需要的安全和隐私漏洞保护的能力。

参考文献

1. Kaner, C. (2008, April). *A Tutorial in Exploratory Testing*, p. 36. Available at <http://www.kaner.com/pdfs/QAExploring.pdf>.
2. Chmielewski, M., Clift, N., Fonrobert, S., and Ostwald, T. (2007, November). "MSDN Magazine: Find and Fix Vulnerabilities Before Your Application Ships." Available at <http://msdn.microsoft.com/en-us/magazine/cc163312.aspx>.
3. Microsoft Corporation (2012). *How To: Perform a Security Code Review for Managed Code (.NET Framework 2.0)*. Available at <http://msdn.microsoft.com/en-us/library/ff649315.aspx>.
4. Ibid.
5. Jackson, W. (2009, February). GCN—Technology, Tools and Tactics for Public Sector IT: "Static vs. Dynamic Code Analysis: Advantages and Disadvantages." Available at <http://gcn.com/Articles/2009/02/09/Static-vs-dynamic-code-analysis.aspx?p=1>.
6. Cornell, D. (2008, January). OWASP San Antonio Presentation:

- "Static Analysis Techniques for Testing Application Security." Available at http://www.denimgroup.com/media/pdfs/DenimGroup_StaticAnalysisTechniquesForTestingApplicationSecurity_OWASPSanAntonio_20080131.pdf.
7. Jackson, W. (2009, February). GCN—Technology, Tools and Tactics for Public Sector IT: "Static vs. Dynamic Code Analysis: Advantages and Disadvantages." Available at <http://gcn.com/Articles/2009/02/09/Static-vs-dynamic-code-analysis.aspx?p=1>.
 8. Cornell, D. (2008, January). OWASP San Antonio Presentation: "Static Analysis Techniques for Testing Application Security." Available at http://www.denimgroup.com/media/pdfs/DenimGroup_StaticAnalysisTechniquesForTestingApplicationSecurity_OWASPSanAntonio_20080131.pdf.
 9. Jackson, W. (2009, February). GCN—Technology, Tools and Tactics for Public Sector IT: "Static vs. Dynamic Code Analysis: Advantages and Disadvantages." Available at <http://gcn.com/Articles/2009/02/09/Static-vs-dynamic-code-analysis.aspx?p=1>.
 10. Cornell, D. (2008, January). OWASP San Antonio Presentation: "Static Analysis Techniques for Testing Application Security." Available at http://www.denimgroup.com/media/pdfs/DenimGroup_StaticAnalysisTechniquesForTestingApplicationSecurity_OWASPSanAntonio_20080131.pdf.
 11. Jackson, W. (2009, February). GCN—Technology, Tools and Tactics for Public Sector IT: "Static vs. Dynamic Code Analysis: Advantages and Disadvantages." Available at <http://gcn.com/Articles/2009/02/09/Static-vs-dynamic-code-analysis.aspx?p=1>.
 12. Cornell, D. (2008, January). OWASP San Antonio Presentation: "Static Analysis Techniques for Testing Application Security." Available at http://www.denimgroup.com/media/pdfs/DenimGroup_StaticAnalysisTechniquesForTestingApplicationSecurity_OWASPSanAntonio_20080131.pdf.
 13. The Open Web Application Security Project (OWASP) (2012). "Fuzzing." Available at <https://www.owasp.org/index.php/Fuzzing>.
 14. R2Launch (2012). "Fuzz." Available at <http://www.r2launch.nl/index.php/software-testing/fuzz>.
 15. The Open Web Application Security Project (OWASP) (2012). "Testing Guide Introduction." Available at https://www.owasp.org/index.php/Testing_Guide_Introduction#Manual_Inspections_26_Reviews.
 16. Coverity (2012). Coverity Static Analysis webpage. Retrieved from <http://www.coverity.com/products/static-analysis.html>.
 17. HP (2012). HP Fortify Static Code Analyzer webpage. Retrieved from <http://www.hpenterprisesecurity.com/products/hp-fortify-software-security-center/hp-fortify-static-code-analyzer>.
 18. IBM (2012). IBM Security AppScan Source webpage. Retrieved from <http://www-01.ibm.com/software/rational/products/appscan/source>.
 19. Klocwork (2012). Klocwork webpage. Retrieved from http://www.klocwork.com/?utm_source=PPC-Google&utm_medium=text&utm_campaign=Search-Klocwork&_kk=klocwork&gclid=CMY0_q6svbICFUjhQgodOGwAFg.
 20. Parasoft (2012). Static Analysis webpage. Retrieved from http://www.parasoft.com/jsp/capabilities/static_analysis.jsp?itemId=547.
 21. Veracode (2012). Veracode webpage. Retrieved from <http://www.veracode.com>.

22. The Open Web Application Security Project (OWASP) (2012). "Static Code Analysis." Available at https://www.owasp.org/index.php/Static_Code_Analysis.
23. Howard, M. (2006, July–August). "A Process for Performing Security Code Reviews." *IEEE Security & Privacy*, pp. 74–79.
24. Howard, M. (2004, November). "Mitigate Security Risks by Minimizing the Code You Expose to Untrusted Users." Available at <http://msdn.microsoft.com/msdnmag/issues/04/11/AttackSurface>.
25. OWASP (2013). "Top 10 2013—Top 10." Retrieved from https://www.owasp.org/index.php/Top_10_2013-Top_10.
26. Hewlett-Packard (2012). Webinspect webpage. Retrieved from <http://www.hpenterprisesecurity.com/products/hp-fortify-software-security-center/hp-webinspect>.
27. Hewlett-Packard (2012). QAinspect webpage. Retrieved from <http://www.hpenterprisesecurity.com/products/hp-fortify-software-security-center/hp-gainspect>.
28. IBM (2012). IBM Security AppScan Enterprise webpage. Retrieved from <http://www-01.ibm.com/software/awdtools/appscan/enterprise>.
29. Veracode (2012). Veracode webpage. Retrieved from <http://www.veracode.com>.
30. White Security (2012). "How the WhiteHat Sentinel Services Fit in Software Development Lifecycle." Retrieved from (SDLC)https://www.whitehatsec.com/sentinel_services/SDLC.html.
31. Peng, W., and Wallace, D. (1993, March). NIST Special Publication 500-209, *Software Error Analysis*. Available at <http://hissa.nist.gov/SWERROR>.
32. Ibid.
33. Ibid.
34. Ibid.
35. Ibid.
36. Codenomicon (2012). Codenomicon website. Retrieved from <http://www.codenomicon.com>.
37. Peachfuzzer.com (2012). Peach Fuzzing Platform webpage. Retrieved from <http://peachfuzzer.com/Tools>.
38. Royal, M., and Pokorny, P. (2012, April). Cameron University IT 4444—Capstone: "Dumb Fuzzing in Practice." Available at <http://www.cameron.edu/uploads/8d/e3/8de36a6c024c2be6dff3c34448711075/5.pdf>.
39. Manion, A., and Orlando, M. (2011, May). ICSJWG Presentation: "Fuzz Testing for Dummies." Available at: http://www.us-cert.gov/control_systems/icsjwg/presentations/spring2011/ag_16b_ICSJWG_Spring_2011_Conf_Manion_Orlando.pdf.
40. Royal, M., and Pokorny, P. (2012, April). Cameron University IT 4444—Capstone: "Dumb Fuzzing in Practice." Available at <http://www.cameron.edu/uploads/8d/e3/8de36a6c024c2be6dff3c34448711075/5.pdf>.
41. Ibid.
42. Grembi, J. (2008). *Secure Software Development: A Security Programmer's Guide*. Course Technology, Boston.
43. Meier, J., et al. (2005, October). Microsoft Corporation—MSDN Library: *How To: Perform a Security Code Review for Managed Code (.NET Framework 2.0)*. Available at <http://msdn.microsoft.com/en-us/library/ff649315.aspx>.
44. Ibid.

发布 (A5): SDL 活动与最佳实践

现在你进入了软件开发生命周期的最后阶段，你需要确保软件是安全的，隐私问题已经被解决到软件发布时可以接受的程度。软件安全和隐私需求来自初始阶段并且通过这个周期进行了提炼。本章将带领你进行最后阶段的策略一致检查，最后的漏洞扫描，版本发布前的渗透测试，开源许可检查，以及最后的安全和隐私检查（见图 7-1）。

就像在 SDL 的 (A1) ~ (A4) 阶段讨论的那样，对 SDL 策略的遵守覆盖所有的项目，这些项目包含有意义的安全和隐私风险，并且在每个阶段分析与更新以覆盖新的威胁和活动。在最后的策略遵守检查中，检查 SDL 策略以确保策略支持基于不同开发标准的具体需求，如产品类型、代码类型、平台。

安全漏洞扫描会在你的软件和相关系统中寻找任何遗留的安全漏洞并报告潜在的暴露方式。这个过程通常是自动进行的，由机构内的特定人员进行测试。作为对比，渗透测试实际上利用系统结构的弱点，需要关于测试软件和相关系统领域不同层面的专业知识。一个经验丰富的安全人员或团队提供了第三方独立的观点、高水平的外部专业人员以及“另一双眼睛”来执行渗透测试。

在最后的 SDL 阶段，对软件的安全检查进行评估，所有的安全活动在这个过程中执行，包括威胁模型、工具输出、早期确定的性能需求在这个过程中会被评估，以确定软件产品是否能够发布。我们将讨论这个过程中的三点。

遵守应用程序开源需求以避免代价高昂、花费大量时间的诉讼是很重要的。如果开源软件用作产品或解决方案的一部分，两个需要 SDL 管理人员关注的重要方面是：遵守许可和安全许可。

必须在软件发布前满足隐私需求。隐私需求验证一般是与最终安全审查以及许多在同一阶段考虑的测试用例同步进行的。

7.1 A5 策略一致性分析

就像 SDL 的阶段 (A1) ~ (A4) 讨论的那样，SDL 策略一致性覆盖了所有有价值的的安全和隐私风险，并且在每个阶段都进行分析与更新以覆盖新的威胁和活动。特别是，策略中的活动和标准已经在每个 SDL 阶段更新，从安全事件根本原因的分析中学习合并的课程，适应改变的威胁环境，促进工具与技术升级。在随后的阶段，跟踪 SDL 策略的遵守情况，如果需要也会跟踪有高风险问题的项目。从 SDL 的开始阶段，正式定义需要遵守的 SDL 项目质量授权和需求。这个策略变成了管理 SDL 过程的重要部分，包括：

- 归入标准化 SDL 授权和活动的项目类型
- 定义每个 SDL/SDLC 项目阶段必须遵守的策略和过程
- 设置版本发布必须满足的需求

在最后的策略一致性分析中，需要对策略进行检查从而保证可以支持基于不同开发标准，如产品类型、代码类型和平台的特殊需求。

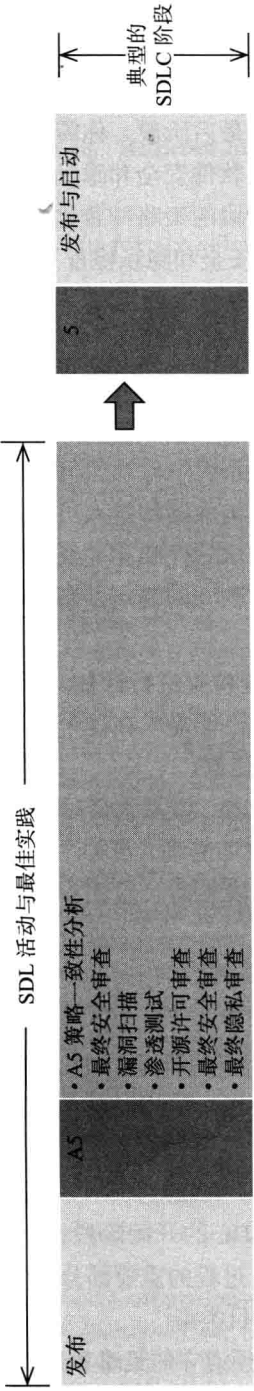


图 7-1 发布 (A5): SDL 最佳实践

7.2 漏洞扫描

尽管现在没有人工审查源代码的替代方法，但是自动化工具依旧有着节省时间和资源的优势。安全漏洞扫描特别适合用于执行这个过程的回归测试，作为双重检查任何可能的安全漏洞，检查因为不注意将之前已经确定并修复的安全漏洞又重新引入代码。也有这样的可能，即其他简单功能的产品在 SDL 的开始阶段有了已经披露的安全漏洞，这些也能够最后的的安全审查中发现。软件产品一般包含 50000 行甚至更多的代码，安全漏洞扫描可以作为性价比高的、节省时间的方法做最后的 SDL 检查。这些扫描器能够执行大规模的复杂数据流分析从而确定人工审查过程中遗漏的安全漏洞点。这些产品通过编译代码库来分析每条可能路径以找到根源级别漏洞，比起人工审查更快更高效。它们也是很好的“检查检查者”工具，即对软件安全架构师已经进行过人工审查的程序进行审查。

安全漏洞扫描工具利用应用程序和数据库签名尝试确认应用程序缺陷的存在。安全漏洞扫描与渗透测试不同，不应归类其中；但是，一些相同的工具可能在两个过程中都会使用。安全漏洞扫描实际上是一个评估，会在软件和相关系统中查找已知的安全漏洞。安全漏洞扫描是自动运行的，技术人员执行后，会报告潜在的问题。开发人员执行安全漏洞扫描可以帮助他们认识和理解软件安全的基本要求。作为对比，渗透测试实际上会暴露系统架构中的弱点，需要软件和测试系统领域各种级别的专业知识。这类测试一般会让经验丰富的软件全专家（例如软件安全架构师或经验丰富的安全工程师）来执行。

安全漏洞扫描是软件安全和 SDL 的必要组成部分。由于它自动化和易于执行的特点，安全漏洞扫描可以作为性价比高、有效性高、最小化不可信因素的方法在 SDL 过程中进行多次持续的测试。对于可能引入新的安全漏洞的过程，如代码或软件架构发生改变后，应该持续对系统进行检测。

尽管所有的精力都用来修复发现的安全漏洞，还是有一些扫描器在发现安全漏洞时出现误报或异常。误报是指，扫描器检测出系统有安全漏洞，但实际上没有。当然，如果经过验证，误报是应该忽略掉的。异常的出现是由于修复会降低最佳的软件性能，限制有风险的功能，或需要对软件架构进行重新设计。风险被认为是可以接受的，因为补救手段能够以最小的开销降低风险。异常可能是持续存在的，或者它们可能有附加的有效期限。典型安全漏洞扫描过程请见图 7-2。

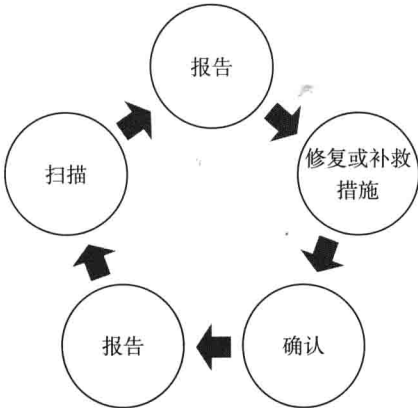


图 7-2 典型的漏洞扫描过程

就像本书前面讨论的那样，静态或动态源代码安全漏洞扫描工具能够在本阶段酌情使用。如果软件是个 Web 应用程序，你必须使用专门的 Web 应用安全漏洞扫描工具进行检测。应避免使用针对所有软件应用的工具，而应使用只针对 OWASP “Top 10” 安全漏洞的 Web 应用安全漏洞扫描工具。就像静态或其他动态安全漏洞扫描工具，如果扫描工具确认了严重的、高风险的应用程序安全漏洞，这些安全漏洞必须在应用程序发布到生产环境前修复。一些通用的 Web 应用安全漏洞扫描工具包括：

- IBM 公司的 AppScan (<http://www-01.ibm.com/software/awdtools/appscan>)
- GFI 公司的 CFI Languard (<http://www.gfi.com/network-security-vulnerability-scanner>)
- Cenxic 公司的 Hailstorm (<http://www.cenxic.com/index.html>)
- McAfee 公司的 McAfee Vulnerability Manager (MVM) (<http://www.mcafee.com/us/products/vulnerability-manager.aspx>)
- Tenable Network Security 公司的 Nessus (www.nessus.org)
- eEye Digital Security 公司的 Retina Web Security Scanner (<http://www.eeye.com/Products/Retina/Web-Security-Scanner.aspx>)
- HP 公司的 WebInspect (<http://www.hpenterprisesecurity.com/products/hp-fortify-software-security-center/hp-webinspect>)

应使用尽量多的安全漏洞扫描器对栈进行扫描。单独使用 Web 应用安全扫描工具是不够的，因为软件栈（操作系统、Web 服务器、应用服务器）也同样会有需要修补的安全漏洞。安全漏洞扫描应该包括外部扫描、内部扫描和对整个栈（特别是云环境）验证的扫描。外部扫描主要在防火墙外发现目标系统的安全问题。由于防火墙通常会限制端口，因此这种扫描有时只能得到有限的信息；但是，它也是非常有价值的，因为从外部扫描中找到的漏洞是可以被攻击者利用的。内部扫描是从防火墙内部执行的，因此找到的信息是不受端口限制且从防火墙外部无法发现的问题。内部扫描可以确定内部攻击者和恶意软件能够利用的安全问题（并且不是防火墙对外限制的问题）。验证扫描是最为全面的检测，不仅仅检测外部和内部的安全问题，还确定漏打的安全补丁和误报。验证扫描需要软件登录到系统上进行扫描，因此这也是最具侵入性的。

在早期阶段，安全架构应该按照软件栈的配置需求加固栈、降低攻击面。配置标准应该以多种形式保留，包括操作系统和其他需要部署的软件的加固标准。例如，一些已经下架的操作系统可能会有很多不需要的服务和配置从而增加了攻击面。加固标准能够有效降低默认配置带来的风险。

除了安全漏洞扫描之外，安全配置应该已经确认以保证栈自身是安全的。理想的解决方案是创建一个栈的“加固镜像”并打上安全审核的标记。产品在操作环境下最终部署后，对这个镜像的任何改变都应该引起注意。

7.3 渗透测试

渗透测试是模拟黑客行为对软件系统进行的白盒安全分析，以发现由代码错误导致的潜在安全漏洞、系统配置错误或其他操作环境下的部署缺陷。渗透测试也会用来验证代码是否按照预期设计实现，验证是否实现了安全功能，并发现可利用的安全漏洞。白盒测试需要了解系统是怎样实现的，从而确保软件产品的健壮性，以对抗预期和非预期的软件行

为，包括恶意攻击和常见的软件错误。白盒测试人员必须有专业技能和丰富的经验，了解是什么导致软件安全和不安全的，如何像攻击者一样思考，如何使用不同测试工具和技术模仿攻击者的行为。渗透测试是通常使用自动化和人工代码审查方法结合起来执行的测试，需要分析数据流、控制流、信息流、代码执行情况、软件和相关系统的异常以及错误处理。

为了成功对正在开发的代码和交互系统执行白盒安全测试，整体上必须满足三个基本需求，而不是独立地满足每个需求。评估者 / 测试人员必须做到以下几点。

- 1. 已经能够理解和分析可用的设计文档、源代码，以及相关开发工件，并有软件安全的背景 and 知识。
- 2. 能够像攻击者一样思考，有能力设计测试用例验证软件漏洞。
- 3. 拥有借助不同工具和技术进行白盒测试的知识与经验，能够跳出局限或非常规地思考，就像使用相同工具和技术的对手。

独立是渗透测试的关键因素和需求，这就是为什么一直考虑要雇用第三方外部安全公司执行安全审查和渗透测试。这么做有旁观者清好处，渗透测试应该在考虑高商业风险后对所有项目授权。外部的观点能够帮助确认安全漏洞的类型，在使用其他方法时无法使现有安全状态完全呈现出来。第三方能够使用威胁模型和在 SDL 过程中创建的架构图，从而确认优先顺序、测试内容以模仿黑客攻击。安全级别要在预算中确定，因为这类公司一般会索要服务的保险费。在渗透测试中确认的任何安全问题或安全漏洞必须被定位并在项目批准发布前修复。

为了获取渗透测试的最小需求，应该遵循图 7-3 中四阶段的流程。

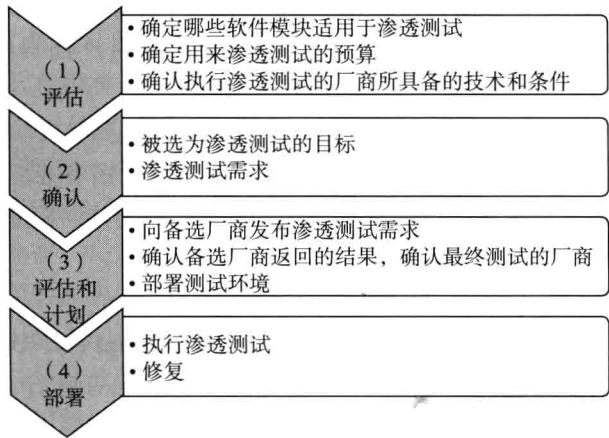


图 7-3 渗透测试的 4 阶段过程

渗透测试报告是渗透测试最终提交的成果。报告的主体应该关注哪些数据是危害的以及为什么，提供给客户现有的攻击方法、漏洞利用方法以及利用数据。如果 SDL 和开发组需要，报告中也应该包含可能的解决方法。已经尝试利用过的和误报的安全漏洞细节列表，或利用但是结果中没有数据的安全漏洞应该在附录而不是报告正文中列出，从而保持报告的主要部分简洁。

安全漏洞扫描生成的可能存在的安全漏洞列表应该在漏洞检测报告中，而不在最终的渗

透测试报告中。就像在之前章节提到的，每个测试活动的目的和结果都是不同的。

考虑到一些非常著名的公司被攻击过，以及知名的产品由于安全漏洞成为新闻的主角，执行安全性测试太迟是导致所有错误的原因。这些导致了客户（传统软件或 SaaS/ 云服务）要求企业级演示以证明其所购买产品 / 服务的安全性。在发布周期的这个时刻上，最好让销售和市场团队一起建立一个与客户讨论安全性的框架。讨论组应该考虑建立一个在销售周期给客户（或潜在客户）提供的安全白皮书。他们也应该考虑为安全白皮书创建一个年度审查周期以对新问题进行确认和修复。客户通常会需要演示安全性是否遵守他们公司的要求，因此会达到公司提供的产品和服务的安全特征。据我们的经验，这种要求通常需要以渗透测试结果或安全细节检测报告（季度或年度）的形式提交。需要注意的是，云或者 SaaS 环境不太适合允许用户灵活地进行渗透测试或共享安全检测结果。尽管不够灵活，但是能够使所有客户达成服务级协议对问题进行修复。简而言之，如果你不希望客户干扰你自己处理安全问题的优先级，与客户形成一个安全讨论的框架式是很重要的。

7.4 开源许可审查

尽管开源软件是免费的、创新的、高效的，并且使得软件产品富有竞争力，但是它也必须按照资产进行管理，遵守许可证，必须像内部开发软件标准一样满足安全需求。有时这些独特和复杂的许可可以延期，并防止发布过程中不适当管理导致的潜在商业风险。遵守开源要求是很重要的，能避免花费很多金钱和时间的起诉。当产品的一部分使用了开源软件时，在 SDL 中管理许可的遵守和安全性上，有两个主要的部分需要关注。

1. **开源软件协议的遵守。**不遵守开源软件许可要求会导致价格高昂和耗时的起诉，开庭时间，侵犯版权，媒体披露，不良的公众形象，不遵守许可会带来风险和消极的商业关系。不当的管理和不遵守开源许可也会导致无法对软件产品提供支持，版本发布延期或停滞。

2. **开源软件安全性。**SDL、开发团队以及投资者需要意识到并理解与开源软件代码相关的安全漏洞会在他们的产品中出现。就像自己开发软件一样，所有开源软件已知的安全漏洞以及在安全社区公布的安全漏洞必须在 SDL 过程中使用相同的威胁模型进行确认，评估以及修复，架构性的安全和隐私审查，风险评估。

一些没能适当地管理开源软件许可或安全性的例子如下。

- **Diebold 和 PES。**Artifex Software，开发开源 Ghostscript PDF 软件的公司，对投票机供应商 Diebold 和它的子公司 Premier Election Solutions 进行了起诉。Artifex 指出 Diebold 将 Ghostscript 合并进了商业电子投票机系统，违反了通用版权许可（GPL）。Ghostscript 最初开发于 20 世纪 80 年代晚期，依据 GNU GPL 进行免费分发。这个许可允许开发者学习，修改，使用，重新发布这个软件，但是衍生品需要遵守相同的条款才能使用。在闭源软件项目中使用 Ghostscript 的公司可以通过从 Artifex 购买商用许可的方式避免版权问题。已经从 Artifex 购买 Ghostscript 许可的商业用户中在印刷技术领域有很多著名的公司，包括 HP、IBM、Kodak、Siemens、SGI、Xerox。^[1]
- **Skype。**Skype 的 Munich 在德国的区域法院被判处违反了 GNU GPL。这个判决影响了公司遵守 GPL 的方法。^[2]
- **Verizon。**两个软件开发者对 Verizon Communications 的诉讼得到了判决，他们声称这

个电信巨头的宽带服务违反了开源协议的许可条款。诉讼集中在转包商在 Verizon 的无线路由器中使用了名为 BusyBox 的开源程序。作为判决的一部分, Verizon 的转包商 Actiontec Electronics 必须向开发者 Erick Andersen 和 Rob Landley 支付一笔未知数额的赔偿。它也必须任命一个内部官员负责确认遵守开源软件许可。^[3]

- **Google**。Google 和其他公司持续获得了负面的宣传, 起因是它们使用的 Android 移动平台发现了大量的安全漏洞并持续成为大量黑客的目标。McAfee 在 2012 年跟踪手机恶意软件显示的数据, 恶意软件从 2011 年到 2012 年增长了 700%。接近 85% 的智能手机恶意软件运行在 Android 上。被攻击的 Android 手机巨大的增长数量还不是最让人吃惊的: Google 的智能手机平台已经成为了攻击者最为关注的目标。最让人吃惊的是, Google 没有采取有效的管理措施阻止这个趋势持续发展。关于 Android 的安全问题已经不是 Google 的新闻了, Google 应该将安全问题放在最高的优先级上。^[4]
- **Oracle**。安全专家指控 Oracle 没有关注它的旗舰数据库软件并且漏报了“根本”的缺陷的严重程度。尽管 Oracle 在 2013 年 1 月的关键补丁更新中修复了很多产品中的大量缺陷, 但是安全专家批评 Oracle 并没有修复多少有着严重问题的旗舰版数据库, 并且在淡化缺陷的严重程度。就像 Oracle 扩展它的产品, 并且通过四分之一的 CPU 增加了进行补丁修复的产品的总数, Oracle 的补丁进程出现了瓶颈。这个 CPU 是 Oracle 第一次包含开源 MySQL 数据库, 在 2010 作为 Sun Microsystems 收购的一部分收购它。
- **CNET Download.com**。CNET Download.com 在它发布的数千个软件包中加入了间谍软件、广告软件等恶意软件, 包括它们自己开发的 Nmap 安全扫描器。他们的行为甚至明显违反了他们自己的反广告协议(它们移除了页面中反广告软件/间谍软件的承诺)。在受到广泛批评后, Download.com 移除了 Nmap 和其他软件中安装的流氓软件, 但是公司依旧进行广泛使用并宣布计划对其进行扩展。由于这些原因, 我们建议彻底避免使用 CNET Download.com。从官方网站或有道德的网站(例如 FileHippo、NiNite 或 Softpedia)下载会更加安全。

使用人工方法检查、收集、监控和鉴定开源软件代码是非常花费时间且低效的, 并且是对开发团队稀缺资源的浪费。自动化工具例如 Black Duck Software (www.blackducksoftware.com) 或 Palamida (www.palamida.com) 是对于在 SDLC 开发过程中降低花费并通过 SDL 管理软件安全性的有效和高效的方法。Black Duck Software 的产品和服务允许机构分析软件源码和二进制文件, 查找复用的代码, 管理开源和第三方代码协议, 遵守混合来源代码的法律责任, 监控相关的安全漏洞。^[7~9] Palamida 能够让机构通过回答问题“你的代码包含什么内容”来管理复杂性持续增长的多来源开发环境。通过对代码库的详细分析, 用户会对代码库的质量控制、风险规避、漏洞评估有更为深刻的认识。^[10]

7.5 最终安全性审查

在最终的软件开发安全审查过程中, 执行的所有安全活动, 包括威胁模型、工具输出, 以及流程早期的性能需求确认, 都进行重新评估以确认软件产品是否为发布做好准备。这个流程会导致以下三种结果之一。

1. **最终安全审查通过。**在这种情况下，所有已经确认的最终安全问题已经修正，软件已经确认满足了所有 SDL 需求，从安全角度来说已经具备了发布的条件。

2. **最终安全审查通过但是有异常。**在这种情况下，所有已经确认的问题并未全部修正，但是作为妥协可以接受开发团队无法处理的一个或多个异常。作为异常，没解决的问题将不在当前版本进行解决并且会被定位在接下来的补丁或版本中进行修正。

3. **最终安全审查没有通过，需要进行扩展。**在这种情况下，SDL 和开发团队对一些安全漏洞与修补情况无法妥协，因此软件不能够发布。一个典型的商业理由是，SDL 过程早期没能发现未遵守 SDL 安全需求的问题。导致软件无法发布的 SDL 需求无法被两个团队解决并且必须扩展至更高的管理层面来决策，用来评估不满足需求进行软件发布带来的风险与后果。扩展至管理层应该查看 SDL 和开发团队提交的包含安全风险描述和理由的报告。

最终安全审查必须仔细地确定时间表，使需要完全分析的时间、修复已知问题的时间、修复最终安全审查发现的安全问题的时间最大化，使软件产品发布的时间更加充裕。

最终安全审查的过程应该包括以下几项。

- **日程安排。**产品安全审查必须设立日程，保证在这个步骤能够获得所有 SDL 需求信息，分配充足的时间从而最小化版本发布的延期时间。开始日期直到确定了所有安全检查活动并且 SDL 过程的开始阶段已经完成后才能确认，包括深度的安全漏洞审查、威胁模型、静态测试、动态测试、模糊测试工具分析。
- **具体的最终安全审查任务。**
 - SDL 和开发团队确保所有在 SDL 过程中已经出现和记录的问题都达到了符合要求的程度。
 - 已经在最终安全审查的开始阶段对开发早期的威胁模型进行了审查和更新，保证所有已知和怀疑的威胁已经被减轻。
 - 所有安全漏洞已经通过了早期建立的标准的检查，至少最小的安全标准已经通过 SDL 获得了增强。任何导致软件产品当前版本拒绝或延期的安全漏洞必须也被审查。需要注意的是，如果 SDL 和开发团队没有在 SDL 过程中持续对安全漏洞的严重性进行评估，会出现大量重复出现的安全漏洞，或者在最终安全审查过程中发现新的漏洞，从而导致不必要的资源和时间占用，并使版本发布延期。
 - 静态、动态和模糊测试工具应该在最终安全审查之前运行，使得在最终版本的结论出来前可以完整地评估结果。在一些情况下，这些工具可能提供不精确或者不可接受的结果，在这种情况下你应该重新运行这些工具或者找到可以替代的工具。
 - 你必须审查和确保遵守了所有相关的内部安全策略和外部规范需求，并且软件也满足了所有的需求。
 - 如果某个 SDL 安全需求无法得到满足，并且整个安全风险是可以接受的，在最终安全审查中应尽早地提出。

最终产品安全审查可以按照 4 步流程进行，见图 7-4。

1. **评估资源的可用性。**在该步中，需要确定执行最终安全审查的资源。增强质量的能力标准应该在软件发布前评估。可接受的最低安全水平应该通过质量标准建立起来。在 SDLC 过程的早期建立质量标准，从而可以使安全风险在 SDL 过程的早期就被认识到，保证了安全漏洞能够在早期被确认和修复，避免了不必要的工作导致的版本发布延期。SDL 和开发团

队必须将遵守质量标准作为最终安全审查的一部分。如果安全已经确认在 SDLC 过程中作为 SDL 的结果建立起来，就可以最小化完成最终安全审查的时间；如果没有建立，就会需要更多的时间和资源，可能会导致版本发布的延期。

2. 确认合格的特征。在该步中，确认最终安全审查的安全任务的合格标准。合格的特征应该在 SDL 过程的早期建立，避免最终安全审查中包含未完成的安全工作。子团队也应该被审查，在 SDL 过程中可能包含没有报告的安全漏洞，在最终安全审查中可能会找到意外发现的遗留高风险安全漏洞。

3. 评估和制订修复计划。在该步中，对任务负责的利益相关者确认之前的步骤中已经得到了通知，对最终安全审查的日程安排进行确认。

4. 版本发布。除了质量标准或漏洞之外，在所有 SDL 需求，例如模糊测试、安全漏洞扫描、安全编码原则检查、其他现有的安全审查得到审查并批准之后，产品安全审查就完成了。功能回归测试也会占用安全审查的资源。回归测试用来发现新软件安全漏洞或根据已经发现的安全漏洞进行回归测试。这些回归测试可能导致已经存在的功能性和非功能性方面的发生改变。简言之，回归测试评估软件的一个部分发生改变是否会导致软件或系统中与它交互的另一个部分发生改变。



图 7-4 最终安全性审查的 4 步流程

7.6 最终隐私性审查

典型情况下，软件的隐私需求必须在发布前得到满足。尽管最终安全审查必须在版本发布前完成，如前所述但是安全例外强调并不是所有安全问题都得到解决才进行版本发布。隐私需求通常可以在最终安全审查的很多测试用例中得到核实。在隐私问卷调查完成后需要做的重要改变，例如收集不同的数据类型，大幅改变语言或风格，或确认新的软件行为可能会对隐私造成潜在的负面影响，都应该被定位。这就需要对软件任何相关的改变，或之前最终安全审查中隐私审查部分确认的问题进行进一步审查。具体的最终隐私需求审查应该包括以下内容。

- 如果项目已经被确认为 P1 项目，那么 SDL 团队和隐私团队必须回顾 Microsoft SDL

Privacy Questionnaire (附录 C) ^[11] 在之前章节提及的其他问卷确认的隐私需求是否得到满足。如果隐私团队认为暴露隐私的问题是可以搁置的，那么就不需要满足这些需求。隐私团队需要批准最终的隐私披露声明发布。

- 如果项目被确定为 P2 项目，那么隐私团队确认隐私设计检查是否有必要，确认软件架构设计遵守软件产品的隐私标准，或确定是否需要例外。隐私团队与 SDL 和开发团队以及法律顾问共同确认公开版本的隐私暴露程度是否适合，保证隐私暴露对于以 Web 为中心的产品是适当发布的。
- 如果项目被确定为 P3 项目，那么没有影响隐私需求遵守的更改被确认，不需要额外的检查或批准，并且最终隐私审查已经完成。如果没有，SDL 团队和隐私团队将会提供一份需要的更改列表。

除了版本发布后响应软件产品安全漏洞需要的责任、流程和程序之外，建立类似于产品功能事件组（PSIRT）的功能以回应版本发布后发现的隐私问题。这部分将会在下一章（关于版本发布后的支持活动）中讨论。

7.7 成功的关键因素

SDL 的第 5 个阶段的成功依赖于策略一致性的最终安全审查、全面的安全漏洞扫描和渗透测试，以及最终安全和隐私审查。表 7-1 列出了这个阶段成功的关键因素。

表 7-1 关键成功因素

关键成功因素	描 述
1. 策略一致性分析	开发阶段中的最终安全审查和一致性需求
2. 安全漏洞扫描	扫描软件栈确认安全问题
3. 渗透测试	利用所有软件栈上的安全问题
4. 开源许可审查	最终审查在栈中使用的开源软件
5. 最终安全审查	最终审查 SDL 周期中确认的安全需求的一致性
6. 最终隐私审查	最终审查 SDL 周期中所有隐私需求的一致性
7. 客户协议框架	确认与客户共享的安全信息的框架

成功因素 1：策略一致性分析

如果确认了任何新的安全需求（基于威胁或更新策略），需要进行在之后开发阶段中执行的可行性分析。一些需求可能不会加入到产品中，而另一些可能非常重要以至于需要延期发布产品直到将这些因素加入进去。

成功因素 2：安全漏洞扫描

安全漏洞扫描以及安全配置验证提供了最后确认和修复软件栈中安全问题的机会。安全漏洞扫描和安全配置验证应该包括不同优势点（外部、内部、身份验证）的评估。它也应该覆盖栈中的所有层，从操作系统到应用程序。

成功因素 3：渗透测试

渗透测试提供了确定安全缺陷是否能够被利用以及会达到什么危害程度的机会。确定安全漏洞扫描和渗透测试没有混淆是很重要的。安全漏洞扫描提供了一个如果被利用会造成潜在影响的安全列表。另一方面，渗透测试是多个安全测试者共同对安全缺陷进行确认的测

试，并且多数以串行方式管理。测试的影响往往取决于技能、想象力，以及渗透测试人员的经验。安全漏洞扫描为渗透测试提供了素材，但这仅仅是一个开始。

成功因素 4：开源许可审查

开源软件的最终审查用来确认所有的需求许可已经满足，对于避免法律责任是很重要的。也能够使不同类型的安全测试工具得到确认（安全漏洞扫描和渗透测试）。

成功因素 5：最终安全审查

在 SDL 的末尾阶段执行的最终安全审查是很关键的。如果所有的需求都满足了，那么安全性就通过了。如果有一些安全需求例外，还需要进行记录和推迟时间。举个例子，一个“条件执行”当不满足需求时不会停止版本发布，但是会在商定的时间内进行修复。

成功因素 6：最终隐私审查

类似于因素 5，这个步骤允许对产品隐私需求的最后审查，在该周期的起始阶段就进行布置。如果没有满足任何需求，它们就应该作为例外记录下来，推迟时间，在确定的时间进行修复。

成功因素 7：客户协议框架

正如前面章节讨论的，一个框架定义为客户参与和安全有关的讨论，不论在销售过程之中以及之后，都是非常重要的。这么做能够限制特殊需求和客户的扩展，让他们觉得你们公司对安全问题都已经掌控从而提升用户的信心。

7.8 可交付成果

表 7-2 列出了 SDL 这个阶段的可交付成果。

表 7-2 A5 阶段的可交付成果

可交付成果	目 标
更新策略一致性分析	分析公司策略
安全测试报告	SDL 的这个阶段找到的不同类型的安全检查结果
修复报告	提供安全修复报告
开源许可审查报告	对开源软件许可需求一致性的检查
最终安全和隐私审查报告	检查安全和隐私需求的一致性
客户协议框架	在产品生命周期的不同阶段对应参与用户详细的框架

更新策略一致性分析

策略一致性分析工件（见第 4、5、6 章）应该在新的需求或策略出现的基础在 SDL 的这个阶段进行更新。

安全测试报告

第 6 章讨论的结果汇总应更新，以包含安全漏洞扫描（外部、内部、身份验证）以及在这个阶段执行的新的渗透测试。应该准备一份分发给企业客户的面向客户报告。

修复报告

除了更新安全测试报告之外，修复报告应该也进行更新，以提升产品的安全性。当前版本发现的没有进行修复的（以及发布前没有修复的）任何安全问题应该被讨论并给出修

复计划。

开源许可审查报告

开源软件的正式审查报告应该准备好，用来证明软件满足了不同许可需求（MIT 许可、GNU 通用公共许可、GNU 宽通用许可、BSD 许可等）。安全和隐私官应该审查并签署这份报告。

最终安全和隐私审查报告

在对安全性和隐私需求进行最终审查之后，需要一份由安全和隐私官签署的正式文件。

用户协议框架

一个与客户分享安全信息的正式归档过程应该在这个阶段完成。这个过程应该包括与客户分享的信息类型（以及频率），需要注意的安全事件，以及发现的安全问题和修复 SLA。

7.9 度量标准

应该在 SDL 的这个阶段进行以下度量。

- 遵守公司策略的百分比（更新的）
 - 阶段 4 与 5 中遵守的百分比
- 安全漏洞扫描和渗透测试发现的安全问题的数量、类型和严重性
 - 不同类型测试中找到的交叠的安全问题
 - 不同类型测试发现的安全问题严重性的对比
 - 发现的安全问题与之前确认的威胁 / 风险的映射
- 发现的安全问题的修复数量（更新的）
 - 发现的问题的严重性
 - 修复发现的问题花费的小时数
- 发现的严重问题的数量、类型、严重性（更新的）
- 遵守安全和隐私需求的百分比

7.10 本章小结

本章对成功的软件产品版本的需求进行了描述，完成了 SDLC 和相关的 SDL 活动以及最佳实践（见图 7-5）。既然我们通过了 SDL 的 A5 阶段并且发布了产品，下一章将对 SDL 的 A6 阶段进行讲解，即 SDL 的版本发布后支持活动（PRSA）阶段的概要。在软件产品发布后，软件安全团队、开发团队、隐私团队在公司版权、法律，以及其他团队的支持下，应对可能出现的安全漏洞或隐私问题进行答复。另外，应该制订一份关于潜在发布版本问题的响应计划。对于外部安全漏洞披露后的响应，这个阶段应该包含新产品，包括云环境的配置、发布后证书、安全架构检查，当前，遗留和 M&A 产品与解决方案基于工具的评估，以及第三方对满足客户需求的已发布软件产品的检查，常规需求或工业标准。

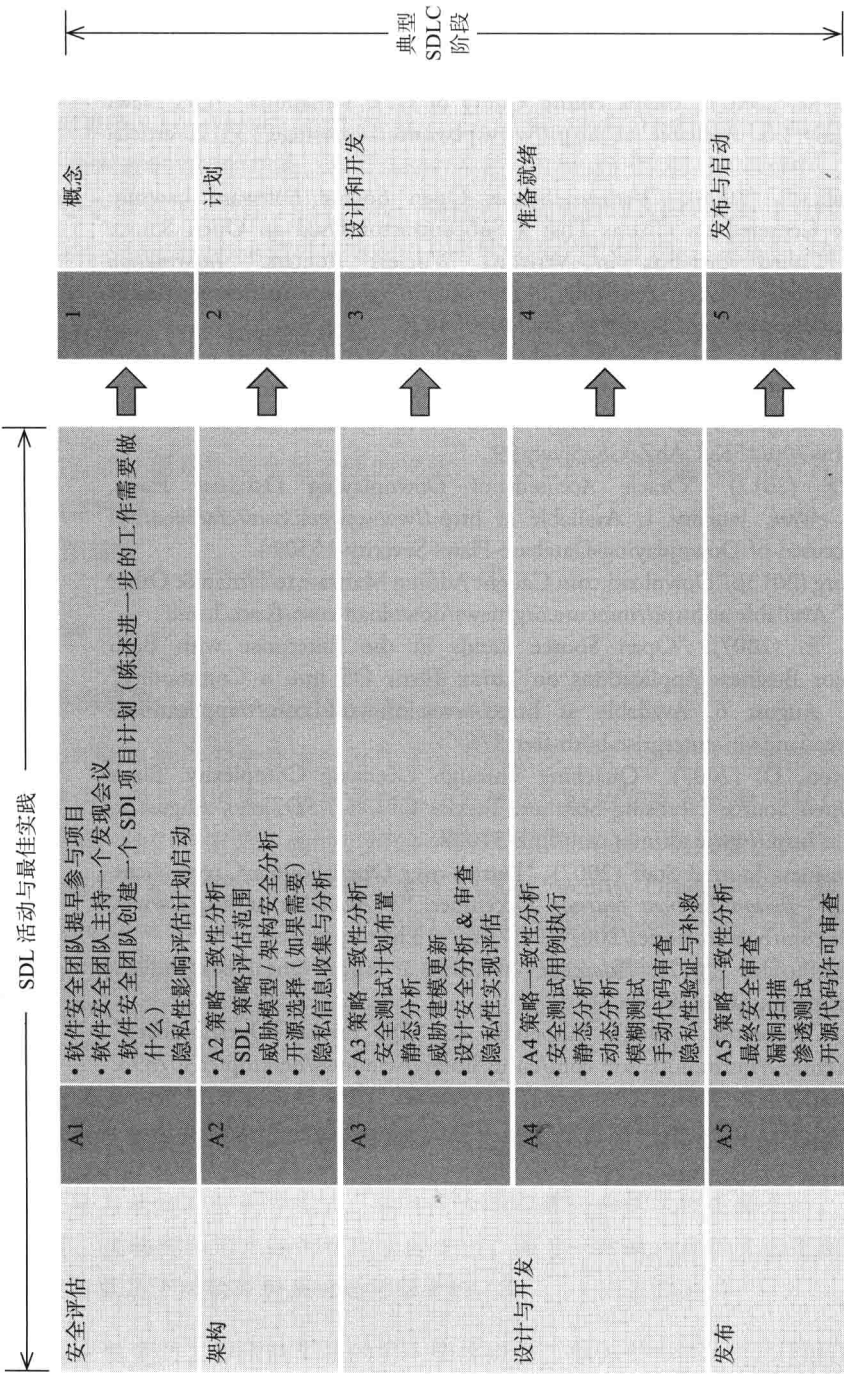


图 7-5 SDL A1 ~ A5 活动和最佳实践

参考文献

1. Paul, R. (2008). "Diebold Faces GPL Infringement Lawsuit over Voting Machines: Artifex Software, the Company Behind Ghostscript, Has Filed a Lawsuit Against. . . ." *Arstechnica: Technology Lab/Information Technology*, November 4. Available at <http://arstechnica.com/information-technology/2008/11/diebold-faces-gpl-infringement-lawsuit-over-voting-machines>.
2. Broersma, M. (2007). "Skype Found Guilty of GPL Violations." IDG News Service, July 26. Available at <http://www.pcworld.com/article/135120/article.html>.
3. McDougall, P. (2008). "Verizon Settles Open Source Software Lawsuit: The Issue Centered on Claims That a Subcontractor Used an Open Source Program Called BusyBox in Verizon's Wireless Routers." *Information Week*, March 17. Available at <http://www.informationweek.com/verizon-settles-open-source-software-law/206904096>.
4. Koetsier, J. (2012). "Sorry, Google Fanboys: Android Security Suffers as Malware Explodes by 700%." *VentureBeat*, September 4. Available at <http://venturebeat.com/2012/09/04/sorry-google-fanboys-android-security-sucks-hard-as-malware-explodes-by-700/#FKvUAhZrG8g5jywy.99>.
5. Rashid, F. (2012). "Oracle Accused of Downplaying Database Flaws, Severity." *eWeek*, January 1. Available at <http://www.eweek.com/c/a/Security/Oracle-Accused-of-Downplaying-Database-Flaws-Severity-155094>.
6. Insecure.org (2013). "Download.com Caught Adding Malware to Nmap & Other Software." Available at <http://insecure.org/news/download-com-fiasco.html>.
7. Schwartz, E. (2007). "Open Source Lands in the Enterprise with Both Feet: Major Business Applications on Linux Turns OS into a Commodity." *Infoworld*, August 6. Available at <http://www.infoworld.com/t/applications/open-source-lands-in-enterprise-both-feet-576>.
8. Worthington, D. (2007). "Quacking Through Licensing Complexity: Black Duck's Open Source Licensing Solution Tackles GPLv3." *SDTimes*, August 6. Available at <http://www.sdtimes.com/link/31007>.
9. Boston Business Journal Staff (2007). "Battles over Open Source Carve Niche for Startup." *Boston Business Journal*, December 17. Available at <http://www.bizjournals.com/boston/stories/2007/12/17/story13.html?page=all>.
10. VentureBeatProfiles (2012). *Palamida*. Available at <http://venturebeatprofiles.com/company/profile/palamida>.
11. Microsoft Corporation (2012). "Appendix C: SDL Privacy Questionnaire." Available at <http://msdn.microsoft.com/en-us/library/windows/desktop/cc307393.aspx>.

发布后支持 (PRSA1 ~ 5)

在本章中 (参见图 8-1), 软件安全组以外的组通过监督前面章节中描述的 SDL 活动和最佳实践 (A1 ~ A5), 对本章中描述的功能、相关活动以及最佳实践进行控制。在本章中, 我们将把它们称为集中软件安全组的责任活动。我们发现, 使用现有资源管理这些活动是更节约成本和有效的方法。这正是我们强烈建议核心软件安全组由高级软件安全架构师组成的原因。在集中软件安全组的安全软件架构师和每个 1 级软件产品的产品经理之间还应该有很强关系, 就像软件安全的冠军。同样重要的是, 该软件的安全组和功能要在正确的组织中, 这样它们可以是最成功的。

8.1 合理调整软件安全组

首先, 我们需要把握每个安全组之间的关系, 以及它们对于构建成功的软件安全程序的重要性。这样做意味着:

- 正确的组织定位
- 正确的人
- 正确的过程

8.1.1 正确的组织定位

尽管在过去几年里软件安全技术已取得了巨大的进步, 但是我们依然坚信人是一个成功的软件安全程序中最重要因素, 其中包括活动和最佳实践的实施和管理。为了让负责软件安全的人人尽其用, 他们一定是正确的组织的一部分 (见图 8-2)。据报道, 本书合著者之一, James Ransome 先生曾经担任过七个首席安全官 (Chief Security Officer, CSO) 和首席信息安全官 (Chief Information Security Officer, CISO)。基于他的工作经验和与业内同行的交流, 很显然, 软件安全功能在理想情况下应该属于工程 (软件开发) 功能, 尤其是质量功能。普遍的共识是, 应用程序安全任职人员通常报告给中心信息安全负责人 (CSO/CISO), 不能混淆软件安全功能。通常情况下, 那些在 IT 安全团队中担任应用程序安全职位的人, 善于使用工具进行测试, 但缺乏必要的软件开发背景来充分解释运行结果。为了搞清楚这一点, 区分软件 and 应用程序安全是至关重要的。也许明晰这种区别的最好方法就是引用 Gary McGraw 的说明:

软件安全与构建安全软件有关: 设计安全的软件; 确保软件是安全的; 培训软件开发人员、架构师和用户如何构建软件安全。而另一方面, 应用安全是在开发完成后真实运行状态下对软件和系统进行保护的。^[1]

软件安全专家负责向工程团队汇报的另一个优点是, 他们是由同一组织授权的; 能够直接负责实施 SDL 政策和程序, 以及相关工具; 能够了解软件开发、架构和修复同一个安全问题所要求的工作量。本书前面介绍了软件安全作为质量和团队一分子的重要性, 这样的关系也应该存在于工程团队中。

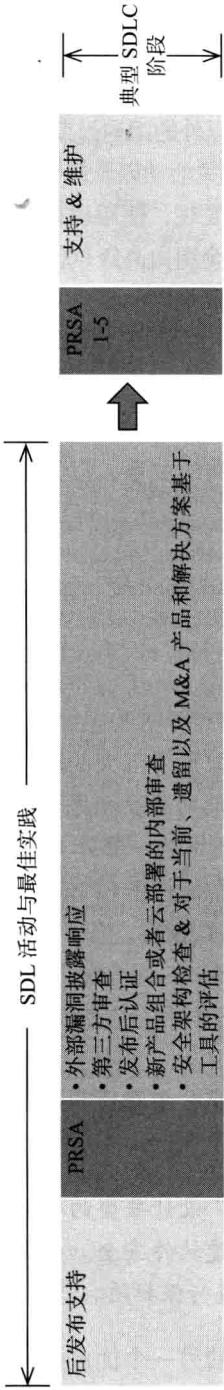


图 8-1 发布后支持 (PRSA1 ~ 5): SDL 最佳实践

笔者认为，软件安全应该是一组工程 / 开发团队，同时与中心安全团队紧密合作；它甚至可能与 CSO/CISO 之间存在“虚线”关系。

软件安全组向软件质量组提交报告的原因有以下几点。

- 1. 按照定义，安全漏洞是质量问题。
- 2. 安全特征是与产品管理关系密切的架构功能。
- 3. 根据上述两点，安全既是一种特征也是一种质量功能。
- 4. 开发过程包括安全方面会使质量管理的内容更加完整，从而达到最佳效果。

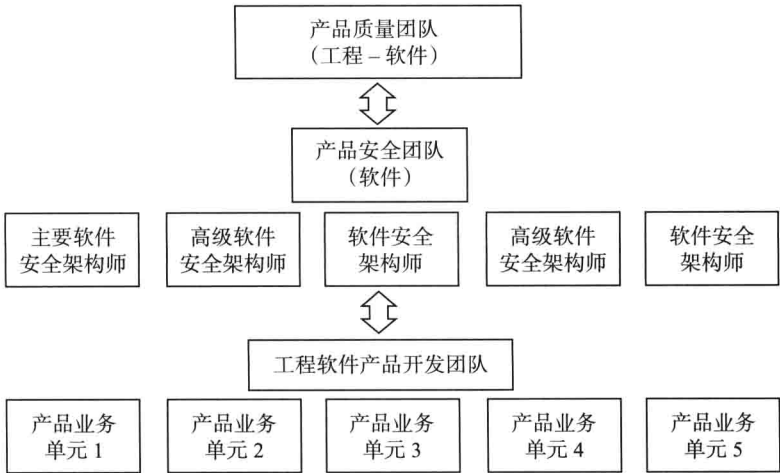


图 8-2 正确的组织定位

8.1.2 正确的人

第 2 章讨论了为了使 SDL 模型成功所需要的人员配置。这包括至少一个主要的软件安全架构师，几个高级和一般的软件安全架构师，理想情况下，在每个软件产品的安全组中，至少拥有一个软件安全架构师。这种关系如图 8-3 所示。这样的人员配置提供了扩展的能力，在每 1 级软件产品的每个工程软件产品开发小组中将有理想的一款软件安全冠军（Software Security Champion, SSC）。人才的另一个要素是组织中的软件产品传道者（Software Security Evangelist, SSE），即组织大到足以有额外的候选人扮演 SSC 的角色，他们在成为 SSC 之前作为 SSE 的候选人存在。SSE 有两个角色，一个是训练中的 SSC，另一个作为传道者的整体软件产品安全程序出台政策、执行政策，以及传道整体的 SDL 流程。

8.1.3 正确的过程

正确的过程就是到目前为止本书描述的和图 8-4 总结的核心 SDL 活动与最佳实践。除了核心活动和最佳实践外，我们在图 8-1 中高亮显示了活动和最佳实践。迫于压力，大多数的公司都要求用最少的资源做最多的事，我估计大多数公司都不会奢侈地将 PRSA 1 ~ 5 的大多数元素作为单独的组织，而是创新性地将其纳入整个软件的安全方案中，以实现现有资源的优化利用。8.2 ~ 8.6 节将会把我们的方法应用到每一个组织成功所需要的活动和最佳实践中。

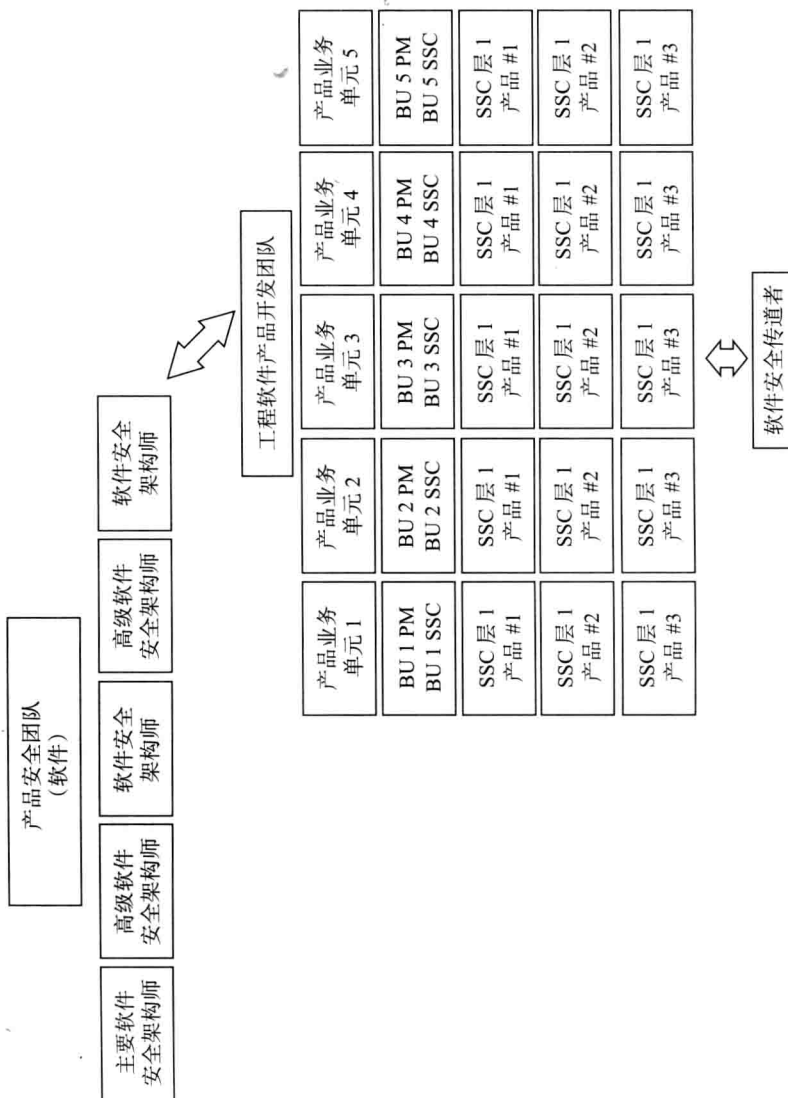


图 8-3 正确的人

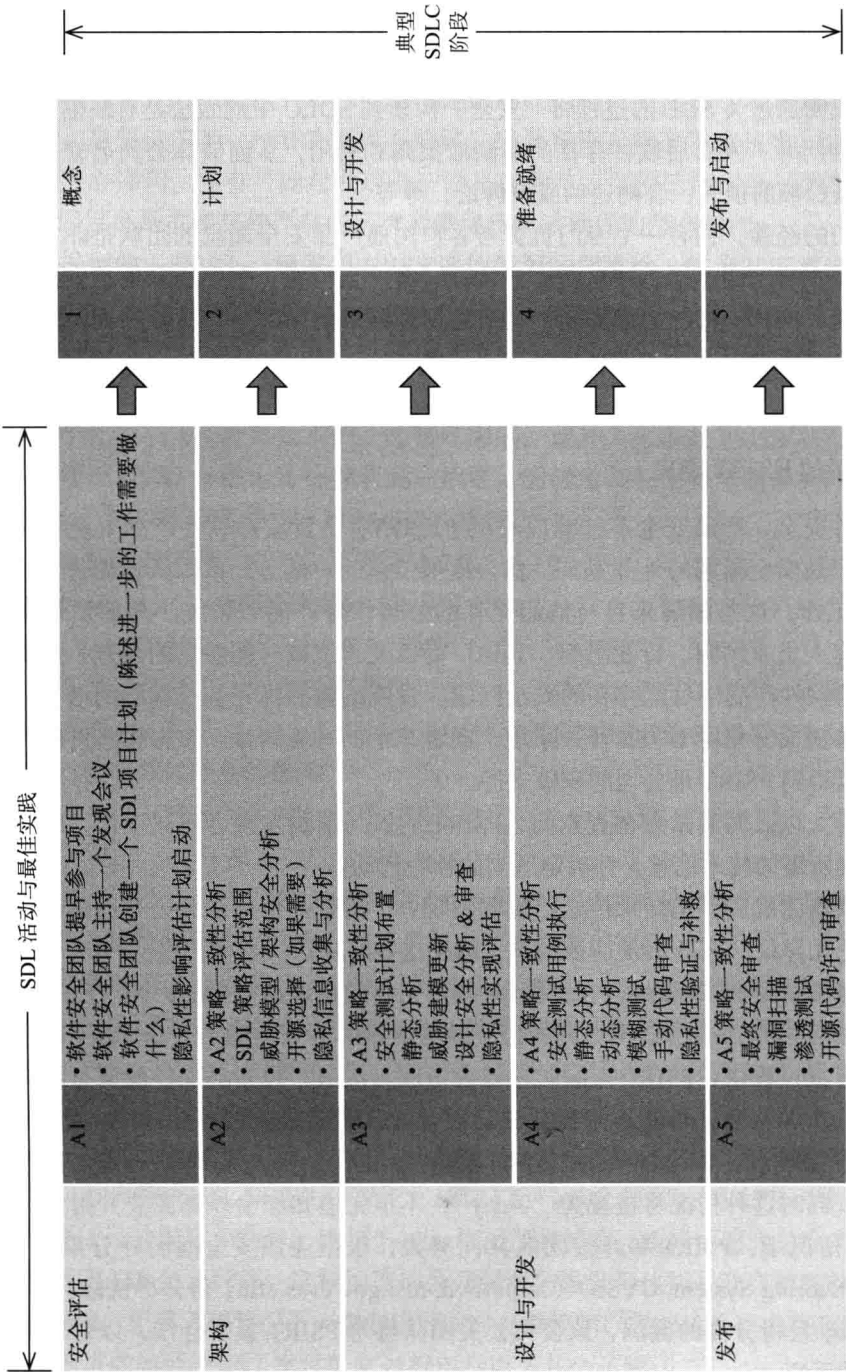


图 8-4 SDL A1 ~ A5 活动和最佳实践

8.2 PRSA1：外部漏洞披露响应

产品发布后的管理关键是要建立产品安全事件响应小组（Product Security Incident Response Team, PSIRT），并与我们推荐的机构在产品发布后发现安全与隐私问题时共同承担责任。无论你的软件在安全方面和相关的 SDL 方面做得多么好，都会忽略一些问题，因此你需要做一个计划以对此作出回应。重要的是，如果在软件发布后经常发现软件安全漏洞和隐私问题，就说明通过类 SDL 的过程将“安全”构建到 SDLC 中的做法是有缺陷的。这样的缺陷会导致各种问题：发布后软件存在的漏洞被披露和利用，从而破坏公司名誉；产品名誉的破坏导致市场份额的损失；合同违约或被诉讼，等等。

根据我们的经验，拥有一个专门负责与客户沟通产品安全漏洞的团队是非常重要的。我们经常看到至少有三组不同的团队与客户沟通：客户支持组、销售组和信息安全组。在一个特定的公司里，PSIRT 可能是也可能不是信息安全组织的一部分，尽管这无疑是可取的。总之，明确规定与客户之间的沟通链是至关重要的，这样可以防止意外信息的泄露，避免因恐慌而使得整个账户危在旦夕。

8.2.1 发布后的 PSIRT 响应

关于软件安全，产品安全事件响应小组（PSIRT）负责应对软件产品安全事件，这涉及发布后软件产品安全漏洞的外部发现。作为这项工作的一部分，该团队管理公开发现的安全漏洞的调查工作，这些漏洞来自与他们交互的公司软件产品和系统。外部发现者可能是独立的安全研究人员、顾问、行业组织、其他厂商以及善意或可能恶意的黑客，他们能够找出 PSIRT 负责的软件产品中可能存在的安全问题。发现的漏洞优先基于漏洞的潜在严重性，通常使用本书中前面介绍的 CVSS 评分系统，或者其他的环境因素。该报告事件的决议可能需要升级产品以得到 PSIRT 母公司的积极支持。

识别后不久以及漏洞索赔调查期间，PSIRT 应该与漏洞发现者协同工作，以确认该漏洞的性质，收集所需的技术信息，并确定适当的补救行动。

当初始调查完成后，把结果发送给漏洞发现者，附带一份决议和公开披露计划。如果事故报告的结论无异议，该 PSIRT 应该尝试去解决这些问题。

将要求漏洞发现者严格保密，直到针对客户完整的决议是可用的，同时已经由 PSIRT 在该公司的网站上发布，通过适当协调的公开披露通常称为安全通告（Security Bulletin, SB）。在调查和预申报过程中，PSIRT 与漏洞发现者协调与沟通，包括对事件的调查情况和文档进行更新。进一步的信息，可能还需要漏洞发现者验证索赔要求和利用该漏洞所使用的方法。同时通知漏洞发现者，如果在 PSIRT 公布之前泄露漏洞，那么发现者将不会获得公司公开披露的信誉，同时这种情况将被视为“0-day”，不事先通知经由外部源公开报道的发现。在 0-day 发现的情况下，PSIRT 和开发团队共同努力，根据通用安全漏洞评分系统（Common Vulnerability Scoring System, CVSS）（<http://nvd.nist.gov/cvss.cfm>）得分尽快修补特定的漏洞。对于一个 0-day 且得分高的漏洞，该公司公关团队将与 PSIRT 紧密合作，以管理潜在的负面新闻和客户的反应。

在一个已报告漏洞的调查阶段，在一个高度保密的基础上，PSIRT 负责协调和管理所有的敏感信息。内部分配仅限于那些确实需要了解并能积极协助漏洞决议的人。

PSIRT 也将与诸如 CERT 协调中心 (CERT Coordination Center, CERT/CC) (<http://www.cert.org/certcc.html>) 等第三方协调中心合作, 管理协调行业披露报告的漏洞对他们负责的软件产品的影响。在某些情况下, 多家厂商将受到影响, 并会参与到与诸如 CERT 等中心的协调响应中。如果一个协调中心牵涉其中, 根据具体情况, PSIRT 可能会代表漏洞发现者与中心联系, 或者帮助他们自己动手完成。

如果产品的第三方组件受到了影响, 这将使得修复过程更加复杂, 因为 PSIRT 将依靠第三方进行整治。更复杂的是, PSIRT 不得不协调, 以及在许多情况下直接通知供应商以确保与第三方协调中心协调, 并有可能与发现者一同直接参与。即使已使用第三方组件, 无论是否拥有它们, 假设也是主要软件产品的所有者最终负责该软件的所有组件。

正如上面提到的, PSIRT 一般采用 CVSS 评估漏洞的严重性, 作为其标准流程的一部分去评估他们的产品中报告的潜在漏洞, 并确定哪些漏洞保证外部和内部报告。

CVSS 模型使用三种不同的度量或者分数, 包括基础、时间和环境的计算, 所有三个分数的总和应该考虑作为最后的 CVSS 分数。这个分数代表了某一个时刻; 它专门针对特定的环境, 并且用于优先响应在外部发现的一个特殊漏洞。此外, 大多数 PSIRT 将会考虑修改最后的分数, 以考虑 CVSS 分数未正确捕获到的因素。当确定某一个特定漏洞的严重性以及是否需要报告它时, PSIRT 通常使用下面的 CVSS 指引。

- 高 (High, H): 临界值, CVSS 基础得分 7.0 ~ 10.0
- 中 (Medium, M): CVSS 基础得分 4.0 ~ 6.9
- 低 (Low, L): CVSS 基础得分 0.1 ~ 3.9^[2]

如果存在涉及产品第三方软件的安全性问题, 且产品由该 PSIRT 负责, 那么根据情况以及第三方部分是否有 CVSS 评分, 该 PSIRT 可能使用组件创建者提供的 CVSS 分数或调整分数, 以反映其对整个软件产品的影响。

当一个或多个以下情况发生时, 公开披露 (包括相关的 CVSS 基础、时间分数和 CVE ID 报告^[3]), 通常是用于一个外部发布后发现的事件。

- 事件响应过程已经完成, 并已确定有足够的软件补丁或其他调整措施存在, 以解决该漏洞。可以发布公开披露的代码修复来解决严重性高的漏洞。
- 漏洞的主动利用已经被观察到, 需要发布安全漏洞公告。该漏洞可能会导致 PSIRT 公司的客户风险增加。这一公告可能包括或者不包括一套完整的补丁程序或其他补救措施。如果可能, 补偿控制都会包含在公开的公告中以提供过渡性的保障, 这将限制披露, 直到公布永久性的修复。
- 0-day 公告或者其他潜在的对于影响到 PSIRT 公司的产品漏洞公众意识的提高是可能的, 这将增加客户的风险。在这些情况下, PSIRT 已与该公司的公关 (PR) 团队密切合作, 以帮助评估公共指标和警告, 比如 Twitter 订阅源和博客, 这样的曝光迫在眉睫, 且将会有提前声明的准备。同样, 这样加速的公开漏洞公告将不包括一套完整的补修程序或其他补救措施, 但是理想情况下, 需要进行临时补偿控制以限制风险被识别的可能性。

一个循序渐进的典型 PSIRT 案件处理过程包括以下几个步骤:

1. 被个体发现者或者组织评估的漏洞通知已被接收。
2. 负责软件产品开发的团队, 以及评估发现者的漏洞索赔所需的资源已确定。
3. 如果索赔是可信的, 将指定一个影响评估, 同时修复的时间表将确定。所需要的工作

量和开发修复的优先级要权衡严重性的公开披露可能性和漏洞的风险。在某些情况下，由于开发团队正在开展其他关键任务，可能需要外部资源。如果索赔是不可信的，需要通过发现者获得更多的信息，以确保该威胁能够在测试环境中准确重现。当测试环境确认后，如果依然不可信，那么将通知发现者公司的调查结果。如果尽管公司已经确定它是不可信的，但发现者依然公开声称该漏洞是可信的，那么 PSIRT 通常与该公司的公关团队合作，发布公司的调查结果以回击发现者。

4. 修正时间表，修复一个确认漏洞所需的资源，以及报告格式（例如，安全公告、知识库文章，或其他形式的公示）的承诺。

5. 当补修程序或其他补救方法已经确定后，通过第 4 步确定的报告格式，同时通知所有的客户修复程序可行的日期。

ISO 29147 和 ISO 30111

关于供应商 PSIRT 正常运行的两个国际标准化组织（International Standardization Organization, ISO）标准于 2013 年年底发布：

- 漏洞披露 ISO 标准（29147）
- 漏洞处理过程 ISO 标准（30111）

下面的信息来源于 Katie Moussouris^①在 2013 年旧金山举行的卡内基梅隆大学计算机紧急响应小组（Computer Emergency Response Team, CERT）供应商会议上做的一个演示^[4]。

ISO 29147 提供相关的指导方针：供应商应对来自外部发现者的漏洞报告的方式；供应商和外部发现者之间沟通的推荐过程。这些发现者本质上可以是善意的，也可能是恶意的。为了使供应商能够优化其应对他们产品中外部发现漏洞的能力，他们应该至少具备：

- 有一个明确的方式接收漏洞报告
- 7 天之内确认收到漏洞报告
- 配合发现者
- 问题报告包含实用信息，至少包括：
 - 一些唯一的标识符
 - 影响的产品
 - 如果漏洞被利用，损害的影响 / 严重程度
 - 如何消除或减轻问题（指导或补丁说明）
 - 如果发现者希望被公开致谢，考虑给予发现者信用咨询

ISO 30111 提供相关的指导方针：供应商如何进行调查、分类并解决所有潜在的安全漏洞？是通过外部发现者还是经由供应商的内部测试来发布？为了让供应商优化他们的能力，能够响应他们的产品中发现的漏洞，他们应该至少具备：

- 有一个流程和组织结构，以支持漏洞调查和修复
 - 执行根源分析
 - 权衡各种补救选项以适应现实世界的危险因素，平衡速度与彻底性
 - 如何合适，尝试与其他厂商合作，比如对于涉及多个供应商或者供应链的问题
- 建议处理漏洞报告的一个五步法详细描述如下：

① 微软的高级完全策略主管兼标准制定者。——译者注

1. 收到漏洞报告
2. 认证
3. 解决方案开发
4. 发布
5. 发布后

整个过程类似于外部发现者或者内部测试发现的漏洞，但风险可能会有所不同。如果涉及外部发现者，应该遵循 ISO 29147，重要的是：

- 了解沟通的期望
- 要考虑到解决开发阶段发现者的意图和发布计划
- 通过咨询发布修复，如在 ISO 29147 中定义的流程概述

8.2.2 发布后的隐私响应

除了可以被发现并公开的发布后安全问题外，潜在的隐私问题也应该被发现。根据我们的经验，与隐私相关的问题没有得到与安全漏洞同样的重视，也没有专门一个团队处理这类问题。一家软件开发公司可能有一个首席隐私官（CPO）或同等类型的职位，如一个专门的律师，但大多数公司没有一个相关的工作人员，并有可能限制一个隐私支持专家。这需要在 PSIRT 功能和中心软件安全团队，以及公司的隐私功能之间保持密切配合和工作关系，无论后者是内部的还是外包的。发布后的隐私响应应该融入到 PSIRT 过程中，就像安全应该融入到 SDLC 中一样。考虑到隐私问题或隐私控制漏洞利用潜在的法律性质，隐私顾问应该为响应团队构建基本的谈话要点、响应程序和法律要求升级，以回应发布后发现的任何潜在的隐私问题。一些基本的指导原则如下。

- 隐私专家应该直接参与到隶属于本书前面所描述的 P1 和 P2 类别的所有事件中。
- 额外的开发、质量保证，以及适用于发布后潜在隐私问题发现的安全资源，应该在要参加发布后隐私事件响应问题的 SDL 过程中被确定。
- 软件开发组织应该制定他们自己的隐私应急预案，或者修改微软的 SDL 隐私升级应对框架（Appendix K）^[5] 为自己所用。这应包括风险评估、详细的诊断、短期和长期行动计划以及实施行动计划。正如上面提到的 PSIRT 响应，该响应可能包括创建一个补丁或者其他风险补救程序、回答媒介查询和深入到外部发现者。

8.2.3 优化发布后的第三方响应

不同的团队和利益相关者之间的合作，提供了发布后响应中成功的最佳机会。相比较于仅仅通过专职的队伍去处理发布后 PSIRT 和隐私支持，软件安全冠军、软件安全传道者的集合体，以及软件开发产品经理和质量团队之间正在进行的正规的软件安全计划，以支持和与本书提到的中心软件安全团队合作，能够提供几个显著的优点。

- 直接的 PSIRT 和隐私响应所有权，通过嵌入这些功能到直接负责修复产品的工程设计和开发团队中来实现，这些产品直接受到发现的漏洞或隐私问题的影响。
- 直接了解代码、架构，以及整个软件产品设计和与修复过程有直接影响的功能，在缺乏产品直接知识的情况下，将会通过外部组织实体来提高效率、控制和响应。从本质上讲，这消除了中间过程，简化了流程。

- 这个过程在投资方面提供了更好的回报：通过资源的杠杆作用，获得 PSIRT 和隐私响应功能；通过直接参与和掌控开发团队，获得软件产品源头上的直接知识。
- 直接授权的开发团队和项目经理，他们更直接地掌控修复过程，并将专职软件安全团队嵌入到工程 / 软件开发团队中，这将为响应提供单一的组织责任。
- 软件安全冠军和软件安全布道者控制本地的软件产品经理和相应的产品开发资源，直接驱动由外部发现者公布的漏洞的评估和修复（如果需要的话）。
- 所有上述结果以更快的时间来执行和响应，更重要的是，有利于加快负面新闻的曝光和客户风险的缓解。我们相信提出的组织基础设施，在提供具有成本效益、较少的资源，以及高效的方式来为此类事件的响应方面具有优势，同时将减少专用于软件本身的开发资源的负担。

8.3 PRSA2：第三方审查

在过去的几年中，软件供应商的客户不断要求进行独立审计，以验证或者评估购买的软件应用程序的安全和质量。在过去的几年里，软件漏洞已经越来越多地依赖于高调的数据泄露事件，并导致了更多的客户需要独立并且明显的证据，以证明他们所购买的软件是安全的。当然，这已经给开发软件的公司带来了巨大的压力，以确保安全的软件开发过程已嵌入到 SDLC 中，避免发布后发现漏洞的昂贵支出——这通常是不成熟的、无效的或不存在的软件安全计划的迹象。拥有安全漏洞和隐私问题的发布后代码，应该在开发过程中就已经被发现。由于这种优势，无论公司生产的软件在代码安全方面声誉如何，发布后或者接近发布代码的第三方评估已经成为行业的常态。在某些情况下，它是潜在的或现有的客户要求的。而在其他情况下，它是由公司生产的代码主动进行的。

即使是拥有优秀的软件安全计划的公司，经过较长的一段时间后，由于各种原因软件应用程序会与政策和监管要求有出入。例如，在应用程序的新版本中的新功能或用例，可能会引入新的漏洞或攻击方式，从而导致应用程序不再达标。此外，这些要求可以随时间改变。许多公司使用第三方代码审查，以帮助识别这些情况，而不是浪费其内部团队有限的资源。

第三方测试应包括整个栈的测试，而不仅仅是你的产品。这意味着需要执行前面章节提到的测试以及后续的发布后测试。至少，发布后测试应包括每年的渗透测试（应用程序和软件堆）。初始版本之后发布的任何新代码，应遵循前面章节所述的 SDL 要求。

做到这一点最大的挑战是要用一种及时和具有成本效益的方式，同时在过程中保护源代码和其他知识产权。第三方测试的一些选择包括以下内容。

1. 提交源代码给第三方进行检查。对于那些想要保护他们源代码的软件开发组织来说，这不是一个真正的选择，他们视源代码为最宝贵的知识产权。
2. 对于每一个新版本，与手动渗透测试服务合作，他们亦可以做深潜代码和软件架构设计评估。为了避免源代码脱离负责开发它的公司控制的风险，合约商必须按要求在受控环境的现场工作，遵守特殊的保密协议与具体的指导方针。这些通常包括源代码保护政策和知识产权保护的指导方针。对此的一种替换方法是雇佣一个使用特定工具的公司，该工具仅仅需要披露二进制代码。在这种情况下，合约商像其受到攻击一样检查应用程序、二进制代码，并能确保检测到所有的威胁。这种类型的测试可以用于现场或者远程方式的服务。
3. 购买、安装和培训开发团队使用内部部署的工具和功能，成为较低级的软件安全架构

师，作为软件安全组的扩展来完成软件安全架构评估“人性化的一面”。然后邀请核算师进入你的组织以记录你的过程。许多成熟的软件安全组织已经这样做了。一个成熟的软件安全计划，如在本书中提到的，将有助于形成规模并减少做好这项工作需要的额外人力。把这个内置到你的 SDL/SDLC 过程中，是一个高性价比、高效和可管理的方式。

4. 要求代码的第三方供应商在你的应用程序中做同样的操作。在当今的软件开发环境下，多数的软件开发组织利用在其他地方开发的代码，要么是商用现成产品 (Commercial Off-The-Shelf, COTS)，要么是开放源码的软件。就像使用内部开发的软件，第三方也应该依据每个软件应用程序所有者的要求准备一份鉴定报告。这份报告可能包括攻击面评估、加密评估、架构风险分析、技术与特定的安全测试、二进制分析 (如果源代码不可用)、源代码分析 (如果源代码可用)，以及除了一般的笔测试程序外的模糊测试。

8.4 PRSA3: 发布后认证

有许多关注安全的认证，一个软件开发团队可能会在产品发布后遇到这些认证。基于各种各样的原因，产品发布后认证的添加被视为一个需求，而不是在开发过程中。这些原因可能包括：设计和开发过程中未规划的工业或政府部门中的软件使用；刚开始使用该软件；新的政府、国家、区域、企业或工业部门，以及在软件发布之前未存在的监管要求。在软件开发之前未存在的发布后认证要求是一个可原谅的错误，但是缺少当前需要的任何认证并在 SDL 早期缺少认证都是不可饶恕的。为了避免正在开发的软件用法不兼容，需要公司的内部资源致力于遵从软件使用认证和其他方面的要求，包括隐私要求，也可以是个人或组织专门提供该领域的体验。随着这些类型的认证和要求的数量在全球范围内急剧增加，这将特别具有挑战性。以下是安全或隐私认证或者标准方面例子的一个简短清单，由于市场或用例的变化，符合发布后要求的软件产品可能会成为必然。

- 联邦信息安全管理法案 (Federal Information Security Management Act, FISMA) ^[6]
- 联邦信息标准 140-2 (Federal Information Standard 140-2, FIPS 14-2) —— 加密模块的安全需求 ^[7]
- 美国国防部信息保障认证认可流程 (The U.S. Department of Defense Information Assurance Certification and Accreditation Process, DIACAP) ^[8,9]
- 健康保险流通与责任 1996 法案 (The Health Insurance Portability and Accountability Act, HIPAA)(隐私和安全规则) ^[10]
- 安全港 (隐私) ^[11]
- 面向技术和出口管制认证的联邦服务 (The Federal Service for Technical and Export Control, FSTEK, 俄罗斯)(隐私和安全) ^[12,13]

8.5 PRSA4: 新产品组合或云部署的内部审查

在行业中，我们继续遇到误解，即一旦软件已经通过 SDL，就可以以任何方式重新使用该软件的代码。但这个假设是错误的，因为软件产品发布后发生的任何体系结构变化将很可能给前面安全的代码引入新的攻击风险。由于这个原因，当发布后的代码有一个新使用的软件或体系架构变化时，软件代码必须再一次通过 SDL 过程。任何新的代码也必须通过前几章概述的各类安全性测试。

8.6 PRSA5: 安全架构审查和基于工具评估当前、遗留以及并购的产品和解决方案

8.6.1 遗留代码

虽然它们可能曾经被视为不必要的成本负担,但是我们在 SDL 中强调的最佳活动和最佳实践是发现后的结果,即安全并不总是软件开发过程中的关键因素,有时会导致安全漏洞,以及与待开发软件的初始成本相媲美的风险缓解成本。遗留代码的验收基于所预期发生的假设,即必须证明该软件在功能上是正确的、在操作上是可行的。然而,当涉及软件安全时,意想不到的事通常会导致漏洞。这些安全漏洞不仅在费用上是不可接受的,而且从操作性、功能性和整体风险的角度来看也是不能被接受的。当该软件支持嵌入式关键系统和应用程序时这一点特别真实,如在国家和地区基础设施、交通运输、国防、医药和金融中发现的安全漏洞。在这些应用中,与意想不到的安全软件和系统漏洞相关的责任、成本、使命和业务影响,被认为是不可接受的。除了遗留软件的架构可以被正确评估,并从安全角度分析,否则变化的影响无法预测,也不能更有效地应用。这就是为什么在 SDL 中随后的同样测试和严格审查必须在遗留代码审查中随后进行:作为减轻意想不到错误的手段。如果按照合适的流程严谨来做,这将在确保安全的代码实现方面意义深远,这在遗留和新的代码之间是一致的。

考虑到更换或重新设计的成本,我们会继续使用一个遗留的软件应用程序,尽管其往往竞争力较差且不易兼容新的软件。在这方面,最显著的问题是,该组织有可能已经依赖这个遗留的软件应用程序一段时间,它预期我们在 SDL 中描述的软件开发安全活动和当前推动这些做法的任务。此外,可能需要大量的资金和资源来消除这个安全方面的“技术负债”。技术负债描述的是交付什么和应当应付什么之间的差异。使用遗留代码和技术负债的重要性是大多数公司开发软件的关键。^[14]

具有技术负债的遗留代码也可以存在,因为即使产品应该处在“生命结束”状态,一个或多个客户不会或无法升级到软件的新版本,并且用户恰好是一个重要客户,他认为这个产品对于其业务来说是必不可少的。这个“重要客户”的身份往往导致遗留代码和产品依然在服务中,所以与一个或多个客户仍在使用该产品的关系不受到损害。

因为它是你的财务负债,所以偿还你的技术负债并不总是有必要的。可能会有部分代码应该是固定的,但是软件产品依然如广告宣传的那样工作;优化代码和消除已知的技术负债可能不会产生有价值的投资回报。你也可以决定只选取程序中的代码,因为它不再有用。在这样的情况下,你可能不再需要偿还技术负债。

上述讨论中最重要的是,遗留软件中的技术负债可能包含安全漏洞。在开发项目的过程中,对于一个组织来说,很容易在软件质量和安全性方面变得松懈。最常见的是,当团队在一个给定的时间内预期完成过多的功能,或者质量和安全根本不考虑软件的高优先级特性时,会出现这样的结果。^[15]在这些情况下,在遗留代码中有可能存在安全漏洞,即技术负债的结果。从软件产品安全的角度看,查看遗留代码时主要的任务是权衡解决安全技术负债带来的投资回报和置之不理的危险。两个主要的决定必须要考虑:

1. 你编写了多少 SDL 可能擦除的新代码以取代现有的旧代码? 旧代码量将以什么速度被替换? 剩下的代码面临的安全风险有哪些?
2. 回顾旧代码是一个缓慢而乏味的过程。重要的投资回报决策必须做出。你必须保留这

项工作的资源，以降低现有资源的技术安全负债。这项工作的工作量将取决于在代码开发过程中 SDL 是否存在。如果在遗留代码开发过程中没有 SDL，工作量将会很高。

下面是评估遗留软件应用程序安全性的基本流程。

- 评估应用程序的业务重要性。该软件应用程序可能已经成功地使用多年，这可能是第一次从安全的角度来审视它。事实上，第一次使用这样的安全级别进行审查是极有可能的。如果发现任何安全漏洞或缺陷，尽管有可能只有一个或两个，减轻它们也将可能需要巨大的工作量和大量的资源。重要的是要确定业务的关键性，以便平衡业务风险与安全隐患和在这些情况下的投资回报。
- 确定对代码非常熟悉的人。由于遗留代码是“老”代码，因此如果组织中的某些人了解该软件，它最有可能最近没有被更新或者有几个更新。此外，它可能已经在古老的语言库之上进行开发，和 / 或缺乏文档或者注释。如果是这种情况，那么下一步是进行软件安全评估，这与 SDL 过程中的做法非常相似。如果遗留软件存在原来的开发商、文档和历史信息，以及一些安全性已内置到该软件中，那么安全评估过程可以缩短到只集中地评估现有的知识空白。
- 也应该询问其他基本问题，比如：
 - 因为一个已知的安全漏洞或者缺陷，这个应用程序是否已经被利用？
 - 它有没有得到解决？如果没有，什么可以做这件事呢？
 - 在软件架构、功能，或使用上是否有一些改变可能已经增加了新的安全漏洞或新的攻击水平？
- 使用 SDL 的关键软件安全评估技术，评估软件的安全性。
- 创建提议，这将告诉企业如何修复软件中的安全漏洞或缺陷（成本 + 时间），或他们应该如何快速地考虑替代它（成本）。如果该软件确定将被替换，将会在过渡期间出现风险。所以在遗留代码被替换之前，你需要确保你知道安全漏洞和缺陷的位置，并制定一个计划，以减轻和限制由于软件的对抗攻击或者利用可能带来的任何损害。
- 在业务方面如果修复的成本是不能被接受的，并且没有客户、行业或法规要求需要该安全漏洞或缺陷是固定的，那么将要求业务部门的高级管理人员和律师签署协议，以接受继续使用遗留软件可能带来的风险，其中前者包括开发软件和软件工程开发组织可能的负责人。

8.6.2 兼并和收购

为了保持竞争力，大多数公司要开发新产品，进入市场并寻求替代品，如兼并和收购 (merger and acquisition, M&A)，这发生在当且仅当他们使用现有的资源无法做到这一点时。发生兼并和收购的原因有很多，但通常是渴望提高竞争力、盈利能力，或者公司及其产品的其他价值。在软件领域，这通常是在你的解决方案集或者产品本身中需要的功能。如果 M&A 的主要重点是目标公司的软件，收购获得的人才将是一个加分项。当 M&A 开始被初步讨论时 M&A 活动宣布开始，经过详尽的调查阶段，并继续把目标公司及 / 或收购的技术整合到母公司中。在这个过程中的工作量和范围，将取决于工作的规模和复杂性。应该指出的是，M&A 并不总是包括目标公司的所有资源。它们可能包括某一软件产品的代码，或者对于收购公司来说是有吸引力的和价值的多个技术或产品。

M&A 中的详尽调查阶段是非常关键的，安全在促使其成功的过程中起到至关重要的作用。如果软件是 M&A 的一部分，将需要安全架构复审和自动化工具的使用。它要么通过使用该潜在收购者的软件安全人员，要么通过第三方来完成，这取决于作为评估一部分的限制以及源代码是否可以进行审核。由于源代码的专有性，大多数目标公司不会允许他们的源代码在 M&A 评估过程中进行审核。因此，需要一个自动化的工具，它能够通过静态二进制分析进行全面的代码审查。这是通过扫描二进制级别编译的代码或者“字节”码，而不是审查源代码完成的，并且通常包括静态、动态和手工技术。

进行 M&A 软件安全评估的最佳清单，可以在表 1 “软件保证 (Software Assurance, SwA) 关注” 和表 2 “专有与开发源 (Proprietary & Open Source, GOTS) 和定制软件的问题” 中看到，它们可参见美国卡内基梅隆大学的 “Software Supply Chain Risk Management & Due-Diligence, Software Assurance Pocket Guide Series: Acquisition & Outsourcing, Volume II, Version 1.2”^[16]，网址是 https://buildsecurityin.us-cert.gov/sites/default/files/DueDiligenceMWV12_01AM090909.pdf。另一个类似的和有用的资源是卡内基梅隆大学软件工程研究所的工作文章，“Adapting the SQUARE Method for Security Requirements Engineering to Acquisition”^[17]，网址是 www.cert.org/.../SQUARE_for_Acquisition_Working_Paper_v2.pdf。SQUARE 代表系统质量要求工程 (Systems Quality Requirements Engineering)。这篇特殊的文章描述了 SQUARE 收购 (A-SQUARE) 过程中的 SQUARE，并适用于不同的收购情况。

在 M&A 软件安全审查过程中，软件安全评估人员应该记住的一些关键因素包括以下内容。

1. M&A 软件安全审查的目的不是着眼于摆脱目标软件的元素，而是评估任何确定的安全隐患可能带来的商业风险。
2. 突出显示任何可能会改变交易性质或从负面影响整合的事物。
3. 寻找任何可能成为潜在大忌的事物。

8.7 成功的关键因素

外部漏洞信息披露响应过程

在 SDL 周期的发布后阶段，拥有一个明确定义和记录的外部漏洞信息披露响应过程是至关重要的。利益相关者应该明确，职责分配和责任分配矩阵 (可靠的、负责任的、可咨询的和知情的 (Responsible, Accountable, Consulted, and Informed, RACI) 矩阵) 应创建。更重要的是，只有一支团队应有责任与客户洽谈漏洞和修复。所有其他团队和利益相关者应该与团队一起工作，并确保没有其他沟通途径或者任何信息选择性地披露给客户。通常情况下，大客户或者企业客户会被给予优惠待遇，并获知中小规模企业无法得到的信息。这不是一个好的安全习惯。漏洞信息或者全部透露给大家，或者全部都不透露。选择性的披露不是一个好主意，喜欢与客户玩保密，在某些情况下可能是非法的或影响什么对于所有客户来说是公平和公正的待遇。

同样重要的是，定义和形式化内部漏洞处理流程为整体漏洞管理和整治方案的一部分。除了产品 / 服务的安全团队和外部研究团队之外，员工和内部客户往往会发现安全问题，并将其传达给产品或者运营团队。需要有一个明确定义的过程，以确保所有相关的安全漏洞被捕获，并完成整治队列。

发布后认证

该产品发布 (或部署在云环境中) 后需要的相关认证，应该已经在 SDL 周期的某一早期

阶段确定。认证的要求应该已经包含在安全和隐私要求中。这将防止认证的合规性审计过程中的任何改造或发现。认证通常需要做年度审计或监督审核。安全团队应该与安全合规团队合作，以确保符合所有相关的控制要求。

第三方安全审查

正如我们已经讨论过的，第三方审查在为最终用户和客户证明“安全”方面往往是至关重要的。厂商的首选名单应该由软件团队创建，而这些厂商应审核他们的技能和能力以便处理敏感信息。由于这些厂商将处理敏感安全信息，因此着重注意他们是否使用全盘加密、安全通信，测试一结束就处置任何客户数据，等等。任何时候，有必要进行安全测试，应选择这些厂商中的一个进行测试。整个软件栈和产品组合的安全测试应至少每年都要进行。

任何架构变化或者代码重用的 SDL 周期

任何架构或者代码改变，或者代码 / 组件重用，应该触发 SDL 活动（尽管并非全部改变可能是必要的，这取决于这些变化的重要性）。

遗留代码、M&A 和 EOL 产品的安全策略和过程

遗留代码很可能将永远不会更新或修改。此外，遗留软件栈也将永远不会被打补丁或升级。旧的 Apache Web 服务器上运行的软件会严重依赖它以及操作系统，因而应用程序本身未改变的情况下它不会升级。遗留代码中发现的任何安全问题将需要很长的时间来修复（如果有的话）。处理遗留代码最好的办法是尽你所能远离它。备选方案包括定义安全过程用于管理遗留代码中的安全漏洞，密切（至少每年一次）监控遗留代码，以及隔离运行遗留代码的产品，使它们对环境的风险最小。

在发布后阶段，M&A 安全评估策略是成功的关键因素之一。正如前面提到的，你可能无法访问源代码，因此评估策略需要考虑其中——也就是说，你可能需要使用二进制文件，而不是源代码。最后，M&A 安全评估应该为被收购软件的整体质量提供输入。如果这个评估没有通过精心思考或正确操作，软件安全组或信息安全组最终可能会很长一段时间处理这些后果。收购软件的缺陷可能会削弱部署在环境中的其他产品的软件状态。

除了处理遗留代码、产品和 M&A 的策略外，为产品 / 发布的最新版本定义行尾计划是很重要的。任何行尾路线图能够引导从此刻开始的安全策略。

8.8 可交付成果

表 8-1 列出了 SDL 这个阶段的可交付成果。

表 8-1 主要可交付成果

可交付成果	描 述
外部漏洞信息披露响应过程	定义安全漏洞评估和沟通的过程
发布后认证	来自外部各方的认证，展示产品 / 服务的安全状态
第三方安全审查	除内部测试团队以外的团队进行的安全评估
遗留代码、M&A 和 EOL 计划的安全策略与过程	减轻遗留代码和 M&As 带来的安全风险的策略

外部漏洞信息披露响应过程

该交付成果应明确该过程中的利益相关者，并为他们扮演的角色创建一个 RACI。此外，与客户的沟通节奏应该正式发布，以让公司中的每一个人意识到这一点，当需要时亦

能够调用它。最重要的是，这个过程每次都应该遵循，无论安全是来自外部渠道还是需要披露给客户。

发布后认证

发布后认证可能包括：基于目标市场的多个可交付成果或证书、监管需要和客户需求。这些因素中的任何一个都可以驱动认证策略。如果这些认证的驱动程序仍然存在，认证就应该更新。

第三方安全审查

此可交付成果包括独立第三方的多重安全评估。基于评估至少需要建立两份报告：一份用于内部使用，另一份用于外部使用。外部报告不应该列出安全漏洞的详细信息，或者公开敏感信息。用于内部使用的报告应该尽可能详细列出安全漏洞的相关信息，并且提供短期和长期的整治建议。

遗留代码、M&A 和 EOL 产品的安全策略和过程

在 SDL 下，有三种不同的可交付成果：遗留代码和产品的安全策略、M&A 的安全策略、临终计划。所有这些都应该由利益相关者进行审批，一旦他们已经由每一个人签署就应在实践中实现。

8.9 度量标准

以下度量指标应该作为 SDL 该阶段的一部分：

- 应对外部披露的安全漏洞的时间（以小时为单位）
- 每月全职员工（Full-Time Employee, FTE）处理对外披露过程需要的小时数
- 产品发布后发现的安全问题的数目（按严重程度排列）
- 每月客户报告的安全问题数目
- 任意 SDL 活动中，客户报告的安全问题没有确定的数目

8.10 本章小结

本章结束了循序渐进的 SDL 的概述，涵盖了关于发布后安全和隐私支持的方法，我们认为这些是独特、实用、及时、操作上相关的方法。这种方法不仅将任务和团队的责任融入到 SDL 中，还能够授权中心软件安全团队和工程软件开发团队拥有他们直接负责产品的安全过程。更为重要的是，我们将所需的组织结构、人员和流程融为一体，更加有效且高效，同时最大限度地保证安全的投资回报和发布后环境的隐私支持。在下一章中，我们将采取所有当前讨论到的方法，并使其与各种软件开发方法相关联，无论是瀑布、敏捷、混合，还是介于两者之间。我们已经包括可交付成果和度量指标（第3章到第8章），它们可用于组织管理、优化、衡量软件安全程序的有效性。在第9章中，我们把它们集成在一起以使用 SDL 框架解决现实世界的问题。

参考文献

1. McGraw, G. (2006). *Software Security: Building Security In*. Addison Wesley/Pearson Education, Boston, p. 20.
2. Mell, P., Scarfone, K., and Romanosky, S. (2013). *CVSS: A Complete Guide to the*

- Common Vulnerability Scoring System Version 2*. Retrieved from <http://www.first.org/cvss/cvss-guide.html>.
3. Mitre (2013). *CVE—Common Vulnerabilities and Exposures—The Standard for Information Security Vulnerability Names*. Retrieved from <http://cve.mitre.org/index.html>.
 4. Moussouris, K. (2013). "A Tale of Two Standards: Vulnerability Disclosure (29147) and Vulnerability Handling Processes (30111)." PowerPoint presentation given at the 2013 CERT Vendor Meeting, San Francisco, February 25.
 5. Microsoft Corporation (2013). "Appendix K: SDL Privacy Escalation Response Framework (Sample)." Retrieved from <http://msdn.microsoft.com/en-us/library/windows/desktop/cc307401.aspx>.
 6. U.S. Department of Homeland Security (2013). *Federal Information Security Management Act (FISMA)*. Retrieved from <http://www.dhs.gov/federal-information-security-management-act-fisma>.
 7. National Institute of Standards and Technology (2001). *Federal Information Standard 140-2 (FIPS 14-2)—Security Requirements for Cryptographic Modules*. Retrieved from <http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf>.
 8. U.S. Department of Defense (2013). *Department of Defense Information Assurance Certification and Accreditation Process (DIACAP)*. Retrieved from http://www.prim.osd.mil/Documents/DIACAP_Slick_Sheet.pdf.
 9. U.S. Department of Defense (2003). *Department of Defense Instruction Number 8500.2, February 6, 2003—Information Assurance (IA) Implementation*. Retrieved from <http://www.dtic.mil/whs/directives/corres/pdf/850002p.pdf>.
 10. U.S. Government Printing Office (1996). *Health Insurance Portability and Accountability Act of 1996 (HIPAA)*, Public Law 104-191, 104th Congress. Retrieved from <http://www.gpo.gov/fdsys/pkg/PLAW-104publ191/html/PLAW-104publ191.htm>.
 11. Export.gov (2013). *U.S.-EU & U.S.-Swiss Safe Harbor Frameworks*. Retrieved from <http://export.gov/safeharbor>.
 12. Government of the Russian Federation (2013). *Federal Service for Technical and Export Control of the Russian Federation*. Retrieved from <http://government.ru/eng/power/96>.
 13. Technology Risk Consulting (2010). Russian Federal Technical and Export Control Order of February 5, 2010 No. 58—*Regulations on the Methods and Means of Securing the Personal Data Information Systems*. Retrieved from <http://technology-risk.com/lawproject/FSTEK58.html>.
 14. Mar, K., and James, M. (2010). *CollabNet Whitepaper: Technical Debt and Design Death*. Retrieved from http://www.danube.com/system/files/CollabNet_WP_Technical_Debt_041910.pdf.
 15. Ibid.
 16. United States Government—US CERT (2009). *Software Supply Chain Risk Management & Due-Diligence, Software Assurance Pocket Guide Series: Acquisition & Outsourcing, Volume II Version 1.2*, June 16, 2009. Retrieved from https://buildsecurityin.us-cert.gov/swa/downloads/DueDiligenceMWV12_01AM090909.pdf.
 17. Mead, N. (2010). *Carnegie Mellon Software Engineering Institute Working Paper: Adapting the SQUARE Method for Security Requirements Engineering to Acquisition*. Retrieved from www.cert.org/.../SQUARE_for_Acquisition_Working_Paper_v2.pdf.

将 SDL 框架应用到现实世界中

Brook Schoenfield

在本章，我们想把你们介绍给 Brook Schoenfield。他是一位真正的思想领袖和受尊敬的软件和企业安全架构师。因为 Brook 有作为一位软件安全架构师的丰富经验，所以我们邀请他来写本章。我们相信该主题作为 SDL 最困难和关键的部分，需要一位经验丰富的软件安全架构师的角度提出高可信度的解决方案。Brook 和我们是多年的同事和好友，同时也经历了我们在构建、指导、管理和为大型和小型软件以及企业安全程序提供技术指导时遇到的相同的挑战。接下来的章节浓缩了 Brook 多年的经验，对于在开发期间的安全软件，哪些可行？哪些不可行？更重要的是哪些应当可行？上述问题的答案见本章。本章展示的模型也是 James 和 Brook 几个月来头脑风暴的结果。作为介绍 Brook 的一部分，我们在下面给出其背景。

Brook S. E. Schoenfield 是迈克菲 (McAfee)^① 公司的首席架构师和产品安全主管。他负责给迈克菲公司广泛的产品组合中各个方面的产品安全提供技术支持。先前，他担任欧特克 (Autodesk) 公司的企业安全架构师，负责 IT 安全技术方面的战略。作为思科系统 (Cisco System) 公司高级安全架构师，他是企业软件即服务 (Software as a Service, SaaS) 产品安全的技术主管。Schoenfield 先生已经在多个会议上发表演讲，包括 RSA、Software Professionals、SANS 峰会等，展现他所在领域的专业知识：SaaS 安全、软件安全、信息安全风险、Web 安全、面向服务的架构和身份管理。他已在 SANS 协会^②、思科和 IEEE 上发表多篇文章。

9.1 引言

软件安全依赖于一系列正确执行的任务。没有执行起来能够提供“足够”软件安全的“尚方宝剑”式的任务。安全必须从开发生命周期的早期开始设计。同时每个寻找缺陷的活动都是互补的。遗漏一个活动的同时其他活动仅仅为了补偿这个遗漏就需要付出很多。软件项目之前的区别在于哪些任务提供最安全的投资回报；一些系统的一些活动都是无关紧要的。当一些 SDL 安全任务的应用程序依赖于每一个项目特定的属性时，其他的 SDL 任务隐藏在安全开发的核心当中。这些任务是开发软件的关键，这些软件可以被依赖而且也是自我保护的。不管在什么情形下，该组核心活动适用于任何一个保持安全状态的软件项目。这些活动可以应用到任何一个项目上。

无论采用什么开发方法，都将会有高层次的系统架构任务、软件架构注意事项和软件设计问题。编写生产代码之前，这些任务必须先完成（虽然实验时可以选择跳过其中的一些）。

① 迈克菲是全球最大的专业安全技术公司，McAfee 杀毒软件是全球最畅销的杀毒软件之一。——译者注

② 系统网络安全 (System And Network Security, SANS)。——译者注

在这些代码被测试之后，如果有测试计划要执行，计划执行的测试必须包括功能测试以及从攻击者的角度进行的测试。在这些处理标记（即设计和测试阶段）之间，将会编写代码。生成代码是软件开发的核⼼。在一些方法中，可能有一些设计⼯作仅仅出现在代码编写之前，甚至是在代码已经开发之时。无论采用哪种方法，都会有处于安全开发核⼼的赌注性（table-stake）任务：安全的正确性、同行评审和静态分析（如果有的话）。

这些任务没有一个本身能够构成一个“尚方宝剑”式的活动以提供安全的软件。每个任务相辅相成。创建一个安全和可保护的架构，具有可理解的流和可控的信任边界，使得软件的功能能够被写入到支持安全的环境中。一个精心构思的安全架构应该需要鼓励安全部署和使用的功能。一旦该架构支持必需的安全功能，它就能在一开始就被设计成安全软件，而不是在实现之后再补全。

由于安全编码依然是一门艺术，本地化语言和运行时的变化增加了复杂性，即一个健壮的实际 SDL 操作，也就是“信任但也要验证”。相信开发者能够编写安全的代码。此外，采取多个独立的和互补的保证方法检查代码：同行审查、静态分析、功能测试和输入路径的动态测试。

简而言之，为安全做准备，考虑如何实现必需的功能，构建功能，然后测试代码，保证安全功能如预期的那样发挥作用以及在编码过程中没有引入漏洞。

我们最终坚信：从根本上来说，软件安全是人们必须解决的一个问题；技术只不过是⼈脑的延伸。如我们将看到的，关系是一个成功 SDL 的关键。显然，⼈们设计、编码和测试代码。⼈们必须在做每一件事情时保持安全意识，就是为了使完成的产品具有构成“安全软件”的所有属性。因为每一个 SDL 任务的执行需要聪明、技术娴熟、有创造力的人，所以执行 SDL 的⼈是最重要的组成部分。当我们探讨安全开发生命周期（SDL）的每一部分时，我们将注意到通过巩固关系和提供积极的劳动力可以生产安全的软件。

图 9-1 举例说明了 SDL 的活动流：

架构 => 设计 => 编码 => 测试

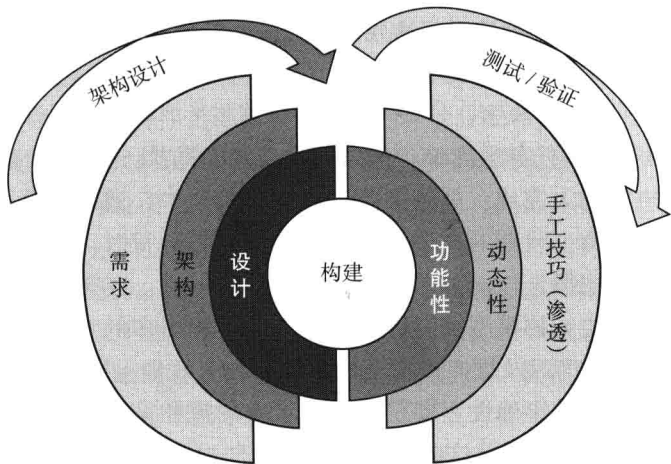


图 9-1 真实的软件开发生命周期

最终，图 9-1 中的 SLD 流减少到一个简单的范例：架构反馈设计，然后构建，即“编

码”。编码设计必须通过一系列的验证，即“测试”。本章详细描述了哪些活动符合图 9-1 中说明的每一个高层阶段，以及这些活动在瀑布或者敏捷开发中应如何使用。当我们从实现的角度讨论 SDL 时，我们将会问关键问题来精确确定哪些活动必须参与到何种类型的软件项目。为了合理地应用安全开发生命周期活动，理解这一点很重要，即不是每一个项目都需要每一个活动。从全新的概念（“未开发区”）或者完全重新设计一个用户接口变化的最小集，应用每一个安全任务到任意的项目中是浪费和昂贵的。要求每个项目，即使没有 Web 服务器，也要经过 Web 漏洞扫描，实在是太傻了。不仅事倍功半，而且开发团队可能认为 SDL 是毫无意义的“文书”，因为它是没有好处、虚空的、形式主义的工作。

过去，熟练的安全专家，通常是安全架构师或者安全工程师，负责从完整的 SDL 菜单中选择合适的任务。随着安全架构师开始了解系统，她（或他）能够规定正确的活动以提供安全软件。例如，知道有一个 Web 服务器只会被部署在一个隔离的网络中，且只由少数几个可信赖的管理人员使用，架构师可能会选择放弃一个详细的 Web 漏洞扫描以支持网络和应用程序的访问控制。必要的安全状态纵深防御可以通过应用多个互补的控件来实现。就像一个画家拥有调色板的技术控制一样，架构师使用他（或他）的调色板构建安全状态。

一些组织能够部署的安全控件不具备高度的灵活性：他们只能部署少数几个；一个完整“调色板”解决方案是不可能的。或者该组织可能不具备将合适的 SDL 任务分配给项目的项目分析技能；或者拥有太少的分析师去实现大规模或多样化的软件开发投资组合。在这些或类似的情况下，一个趋势是采用“通用的”方法。这种方法应用 SDL 的所有任务到每一个项目中，不关心项目大小、目标软件的变更量以及组件的变化。根据我们的经验，这种方法充满了陷阱：抵制、滥用、错过时间表或者团队完全无视安全任务。将安全集成到软件项目中不是一个“通用的”问题。不仅会有浪费，甚至会徒劳无功，而且负责任务的工程师将会对 SDL 失去信心。这将会导致人们对项目的冷漠，甚至玩弄（gaming）这个系统去规避当前开发者面临的问题以使 SDL 成为毫无意义的形式主义。

相反，基于许多不同的组织多年来对成百上千个软件系统成功的安全分析，我们提出了一种简单的方法。这个丰富的经验已经浓缩为一系列关键问题和由答案推断出的任务流。SDL 活动中成功的应用程序能够通过问这几个重要的问题来完成，要么在项目启动时要么在开发过程的每个高级阶段之前。在 SDL 阶段相关联的任务完成之前，每一个定义的问题必须得到回答。

但并不能认为执行这些任务很简单。正如微软威胁建模者已经指出的，威胁建模可以由任何一个拥有好奇想法的人完成。尽管如此，在当前实践中，通常需要一个有经验的架构师，他能够明白项目架构、预期的部署模型、开发语言和运行时。架构师同时必须对于该种威胁代理拥有很强的把握，他们能够积极地应对这类软件。甚至，他必须明白相关的攻击方法，以便构建现实的威胁模型。静态分析工具需要相当多的专业知识，如大多数形式的动态测试、Web 漏洞扫描以及输入模糊测试。学习和有效地运行现有的工具是很重要的。代码审查需要理解一般安全正确性、特定待代码审查的流和结构，以及应该如何实现预期的功能。最初级的工程师可能不是这些任务的最佳人选——至少，得有经验丰富的从业者的重要支持。

如果一个人可以把一系列正确的问题导入到 SDL 中，选择一组正确的高级任务将被证明是相当简单的事情。即使执行的这些任务是重要的，这也是正确的。存在一种依赖关系，即

任务处理流遵循答案之外的逻辑。一旦任务流参与进来，适当的活动将会以一个或多或少线性的方式发生。

显然，使得 SDL 相关和适当，使得 SDL 不同利益相关者之间有意义的交互成为可能。如前所述，让聪明、忙碌的人来做他们无法感知价值的事情，不能加强安全团队的信心。相反，一个透明的过程本质上明显有相反的效果。让 SDL 参与者回答诸如哪些活动适合于他们当前项目的基础问题需要在本质上建立信任。事实是执行如下任务的许多人依然需要深厚的技术（通常还有人际关系）专长。

当前的发展现状没有足够简化一些 SDL 任务，使得“任何人”都能够执行他们，即使开发团队的每一个成员拥有少量的技术技能。对于我们参加的大多数组织来说，体系结构评估或漏洞扫描依然是专家领域。基于这种状况，“关系”变得异常重要。实现安全的软件需要许多人齐心协力朝着共同的目标——显然，这意味着非安全参与者必须明白安全是一个“共同目标”。因此，我们提倡尽可能多的透明度，能够耦合到一个 SDL 实现中，用尽可能多的技术推动人力和关系。关键的决定性问题是现实世界中实现人们关注 SDL 的这一步。

关键的决定性问题将在本章的后面介绍。首先我们将检查每一个 SDL 的核心，那些不依赖于变化数量或者接口类型的活动。如果没有这些核心的任务，SDL 将缺少最重要的组成部分，以保证为了安全编写正确的代码并且包含尽可能少的缺陷：编写安全的代码、审查代码、通过静态分析运行代码。

就像我们检查应用到实际项目中的 SDL 任务一样，本章将详细地描述软件安全程序的支持部分，用来培养和养成需要的技能，以便很好地执行任务。

9.2 安全地构建软件

在安全软件开发的核心，有三个关键的活动。图 9-2 描述了在敏捷和瀑布开发中的这三个关键活动及它们的关系。每一个程序员必须尝试编写安全的、防御性的和自我保护的代码。对于程序员来说，不需要安全路径也可以理解以他们使用的编程语言必须做些什么。不同的编程语言和不同的执行环境需求的侧重点不同。实际上，某一种编程语言存在的问题对于另一种语言可能不值得考虑；程序员仅仅需要了解 C/C++ 和 Java 之间的区别即可理解这个事实。C/C++ 语言允许内存的误处理，事实上，程序员非常容易做一些不安全的操作。相反，Java 语言负责所有的内存处理，程序员不再需要担心内存的分配和回收。

即使编写正确、安全代码的专家也会犯错误。当前的现实是，很少有开发者是安全方面的专家，更不用说某一特定语言和运行时中的安全问题。此外，有一点需要记住，编写软件代码既是一门工程也是一门艺术。当存在一些正确的实现和错误的实现（工程）时，表达正确的实现也需要大量的创造力。对于一个特定的问题可能存在多种算法。任何一种选择的算法将会有多种表达方法。这并没有考虑创新：程序员可能会遇到尚未尝试的计算机问题；他将不得不创建一个全新的算法或者大幅修改某一标准。同时，即使拥有命名规范和其他的编码标准，程序设计语言本质上是富有表达力的。这可能成为程序员的激励因素之一：创造性地工作。但是创造力和创新带来了错误。错误是创新的代价。不是每一个想法都可行，大多想法可能是不可行的。一个非常成功的方法是通过试图解决问题来了解一个问题。这种迭代的方法在构建软件的过程中经常使用，但迭代发现导致了一定程度的失败和错误。我们将假定缺陷和漏洞是创新的直接结果（当然，创新和创造力不是漏洞的唯一原因）。

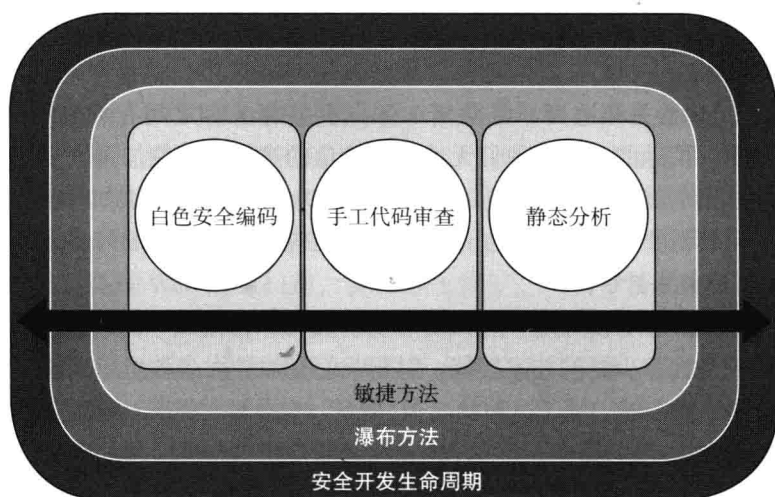


图 9-2 无论使用的是开发方法还是 SDL，软件安全的核心要素是不变的

软件安全从业者面临着一个权衡，即减少创新和降低可能的工作满意度，还是交付代码保证安全性。这就需要保证步骤来提供适当的帮助。我们建议任何一种开发过程的核心中，都应该包括人工代码审查和静态分析。也就是说，给予创造性的程序员大量的帮助以交付安全的代码。

图 9-3 形象地强调了主要的编码流：编写安全的代码、静态分析和人工代码审查。这是在软件构建方面“构建”基本的“安全”版本，也就是安全开发生命周期的“核心”。不管使用的哪种开发过程，这些核心任务都位于安全软件的中心。

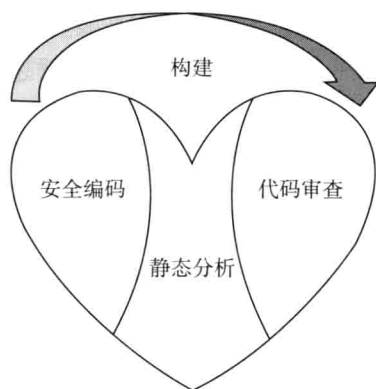


图 9-3 SDL 的核心

9.2.1 编写安全的代码

每个程序有（至少）两个目的：一个是程序要实现什么，另一个是程序不实现什么^[1]。

如果编写软件的软件工程师能够生成正确的、无差错的代码，将会有更少的软件安全问题^①。并且只有正确的软件，就不需要所有的非设计相关的 SDL 任务。在陈述显而易见的风

① 仍然会有逻辑错误、架构和设计遗漏及委托错误。逻辑错误不依赖于正确的代码。很有可能不可靠地指定，然后代码正好在规范那里发生错误。也就是说，它可能正确地遵循不正确的规范然后引入逻辑漏洞。

险，编写正确的代码方面一直是困难的，甚至没有安全方面的考虑。自从软件行业诞生以来，当软件工程师采用术语“bug”（意思是“软件错误”）的时候，工程师们努力使产生的代码是正确的，在逻辑上正确并且没有运行时错误。从所有已经写出的软件的丰富经验推断，我们可以得出这样的结论：编写没有错误的代码是非常、非常困难的。没有错误不是不可能的，但它仍然是一个遥远的理想，而不是一种常态。

认识到编写代码会产生错误是很自然的，软件工程师试图找到一个度量，来显示任何一个特定代码片段的可靠性和无错误性。一个度量方式是每一行代码中缺陷（.bug）的数量。人们普遍认为生产软件大约每 1000 行包含一个代码错误。这是一个非常通用的统计。可能生产的软件含有更少错误；你只关注关键的太空任务或军事软件，这通常必须遵守更严格的缺陷限制。当然，写得很糟糕的软件可能存在超过每 1000 行 1 个缺陷的数量级。然而，通常，在任何适度复杂的代码段中都有错误。今天我们可以说，我们没有办法保证 100% 的软件或安全错误在发布之前被剔除掉。最好的方法是通过所需的 SDL 活动（见下文，确定问题），以确保大部分的错误可以在产品部署或者售卖之前被捕获和修复。如果在软件发布后发现错误，基本上每个人都应该缩短 SDL 周期以矫正错误，测试，发布。

测试显示的是存在的 bug，而不是不存在的 bug^[2]。

除了这些总体上的显著错误之外，一些比例的错误会受影响，这些错误可以被攻击者操纵；也就是说，这些错误会成为安全“漏洞”。我们在这里关注漏洞。

有合理的确定性，开发人员至少有部分时间将产生不正确的代码，并且这些错误将受到恶意操纵。因此，代码中将会有漏洞。这不是一个放弃努力编写正确的代码的理由。一开始就正确编码的好处远远大于其支出；这是一个既定事实，尽早开发周期实现正确性更便宜，更容易。最早的时间点在开始编写代码之时。

有几个互补的活动，当放在一起时，将有助于减少代码编写过程中的安全缺陷：开发人员培训、安全库和证明实现推广以便重用。

显然，开发人员必须知道什么是预期的，哪里需要注意安全，什么是必须编程的正确行为。任何和所有的训练方法都是有好处的，虽然可能显而易见的一些方法的效果不好。

开发人员应该理解，在一个非常高的层面，在他们编写的软件中有些缺陷有着深远的影响。功能规范描述了将如何使用该软件。这些规范之中的安全属性和功能是用户和软件的所有者所要求的。一般情况下，这些已经指定被编写。这并不代表没有错误混进来；确实有错误出现。但由于属性是需求的一部分，因此开发人员有一个有机系统，如身份验证和授权，以及 API 函数和类（实现通信的保护），如 TLS / SSL。如果指定了，开发人员通常会尝试构建这些功能。

安全行业主要对漏洞很感兴趣（可以利用的东西），也就是说，安全缺陷。漏洞是缺陷，操作时会产生副作用（可以被利用）；漏洞是 bug。最终安全从业人员必须评估风险，这是漏洞的一部分，它是一个自然的职业。但是，开发人员不关注漏洞。他们只关注正确性。算法写得正确吗？这段代码实现的规范准确吗？这段逻辑生成正确的路径吗？这段代码对于之前从未见过它的人足够清晰吗？这段代码拒绝无效输入吗？这段代码会崩溃吗？崩溃的项目可以恢复吗？这些类型的问题使开发人员思考。漏洞通常是错误的副产品，通常不是错误本身。

从漏洞到缺陷或错误代码的回溯追踪是在撰写本章时信息安全领域的一个缺失。说到安

全，人们会谈论漏洞。开发人员想知道 bug 在哪里，以及如何改正。开发人员通常问：“什么是必须实现的正确行为？”

把 bug 的作用理解为可以让一个程序充满漏洞对于程序员来说是很有用的。然而，培训应该关注什么是正确的、自我保护的行为。US-CERT 的关键实践（即减轻最恶劣的、可利用的软件缺陷^[3]）将是一个崭新的开始，实践关注正确的、安全的编程。本书的重点不是漏洞的副作用或攻击模式，而是系统特性和算法，它们将防止这些问题。例如，对于跨站点脚本（XSS，也表示为 XXS）错误，现在的 Web 应用程序所特有的，关键实践建议将防止 XSS 与程序解决方案互补起来。下面的引用建议了一种特定的算法，它不仅是“正确”的，同时也防止 XSS[⊖]的变种漏洞。

当可接受的对象集，如文件名或 URL，有限或已知时，创建了从一组固定的输入值（比如数字 id）到实际文件名或 URL 的一个映射，并拒绝所有其他输入。^[4]

注重正确行为的训练是减少安全漏洞的关键。一个安全编码训练计划可能开始于一个安全系统的必要高级属性。有许多安全设计原则集合。开放式 Web 应用程序安全项目（Open Web Application Security Project, OWASP）概括了几个最广为人知的原则：

- 应用纵深防御（全面调节）。
- 使用正面的安全模型（加载安全默认值，减少攻击面）。
- 不安全。
- 最小特权运行。
- 避免含糊的安全性（开放式设计）。
- 保持简单安全（可核查、经济的机制）。
- 检测入侵（有危害的记录）。
- 不要相信基础设施。
- 不要相信服务。
- 建立安全默认值^[5]。

在开发者设计和实现的时候，虽然他们无需进一步的训练就可以透彻地理解这些原则，但是安全的正确性将会是项目初具规模时的关键属性之一。

训练必须是有效的和容易理解的。通常，由于缺少生产力和机会成本，为期一周的课程很难广泛安排。沉浸式的方法当然有好处，但如果不能很快地应用于手头的任务，任何内容都可能会迅速丢失。相反，能够小剂量接受然后联系实践的短片段训练，可能会带来更好的结果。我们发现一种非常有效的模式，结合如下几点建立基础的理解和技能水平：

- 高级应用程序漏洞介绍
- 安全设计和编码原则
- 三到五个要求，易于理解的短期课程（最多为 20 分钟），演示适用于开发人员平台和语言的常见漏洞的正确修复

一旦建立了基础的训练后，再提供常见修复方面的额外训练。这些常见模式的培训必须容易理解。开发者往往很珍惜自己的时间。高度集中的短期课程能够提供软件安全方面的深

⊖ 引用的算法不是特意为了修复 XSS 漏洞，选择它是因为特殊性，而不是因为完整性。

入训练。用这种方式，开发者可能遵循他们自己的兴趣。如果他们必须修复一个特殊的错误，他们可以选择接受短期的集中培训。或者，如果他们掌握了“安全之火”，他们可能想要探讨诸如加密实现或者访问控制系统等更深层次的主题。任何训练计划应该提醒工程技术人员，安全技能是适合市场的重要技能。

没有机会使用学到的技能，培训本身就没那么有效了。当然，熟能生巧。这就是要在培训课程之间给开发者时间以应用新技能的理由。自然节奏的训练后，实践的建立应该结合以下两点。

- 30 ~ 60 分钟的安全编码训练。
- 几周的实际应用。

那些成为专家的人可以帮助那些不具备很多技能的人。更进一步，这些专家成为高级代码审查人，不仅能够在人工代码审查中发现缺陷，还可以传播正确的方法和算法。

经过一个健全的安全编码培训计划，可以实现程序的彻底调试。这些库应该被设计成能兼容各种实现的通用解决方案。这些库应该为库里功能的正确实现提供易于实现的 API。这些正确的库可以包括在每个软件项目中，来确保安全得到有效实施。并且，如果在库里发现了一个错误，这个错误只需要在一处修复而不需要在多处不同的实现中修复。

OWASP 的企业安全 API^[6] 是审查实现的一个例子，它能解决在 Web 应用程序中出现的各种安全问题。例如，防止 XSS，应用中使用的所有输入在传入应用之前用白名单的方式进行验证：

在使用前，所有输入必须用 `Validator.*` 方法进行严格的白名单验证。^[7]

确保代码足够安全的关键技术在于正确的 API 是否被内置或者引用。创建和使用审查过的库将会确保正确的实现会被使用或者重用。这也避免了实现错误的问题蔓延到每个独立程序员的实现过程中。

另外一个可能有帮助的解决方案就是结对编程。Kent Beck 对结对编程的定义如下：

一对程序员在一起编程，他们不仅一起做测试用例，还一起投身到系统设计中。变化并不局限于任何特定的区域。结对编程增加了系统分析、设计、实现、测试的价值。结对编程无处不在地增加了系统所需要的价值。^[8]

用这种方式，无论是设计还是实现都是对结对编程程序员技能的提高。

编写安全正确的代码是几种方法协同工作的产物。培训能补充实践能力。实践收获专业知识，这些专业知识可以用于审查实现，可以重用于把错误降到最少。协作方式工作构建了团队专业知识，在编写代码的过程中能够捕捉到错误。然而，只有安全是不够的，还必须验证正确性。

9.2.2 人工代码审查

由于计算机编程语言的表达特性和多种多样的逻辑处理问题，聘请一个独立、熟练的代码审查员就显得非常重要，它能为程序员提供一个安全网并能检查代码正确性。人工代码审查能轻易发现代码中的逻辑错误。如果审查人知道代码做了什么事情，却没有参与到代码编写中去，他就能经常发现错误，特别是逻辑错误。审查人确定能检查出代码所遵循的标准。

如果审查人有计算机安全方面的专业知识，他也可能发现安全缺陷。因为缺乏安全编码方面的专家，所以可能有必要把焦点放在程序逻辑和代码风格上。然而，如果对每个开发人员进行基本的编程防御培训，特别留意输入数据，人工代码审查至少可以发现令人震惊的输入数据验证错误问题，它困扰了现在的很多软件开发。但在任何情况下，代码审查不是万能的。虽然让专家人工审查代码是可能的，他能从字面上让你写的代码不仅安全而且高效、优雅、易于维护，但是这都是非常昂贵的。很少有这样的“代码大师”。对于关键代码部分来说，使用这些专家也许是很划算的，但在大多数项目中，对于大多数组织它是不可行的。配合多种不同的办法是可行的，例如通过针对性的、广泛的、手工的和自动的方法来保证项目。让这些互补性的方法最大化的方法已被证明可以发现最广泛的缺陷，同时这些方法也是可以伸缩的。

最大化地人工审查代码是十分繁重的，却容易理解。典型的代码必须围绕关键的和专有的业务逻辑模块，对大多数组织的同行审查来说可能已经足够了。人工同行代码审查是在代码提交构建之前由一个或者多个开发者同行审查代码。同一个开发团队的成员可能会熟悉代码的实现目的是什么，代码的运行环境，项目的一般架构。对开发人员来说，理解项目并提出建设性的意见并不困难。对于不太复杂的代码，像可以调用广泛理解的 API 的例程，一个审查人可能就足够了。对于更复杂的代码，可能需要多个审查人。事实上，一个强健的同行审查程序的隐形好处就是，增进了团队成员之间的相互信任。此外，审查人将会成为程序员的后备人员，这样就没有一个人是必不可少了的。在一个团队里，程序员就会互相信任，以至于他们往往会要求对他们的代码进行多次审查。结果就是代码比团队代码审查之前更正确、更高效。如果团队足够熟练，同行审查甚至可以产生关键的或者高度敏感的算法。

然而，通常，特别是在一个快速发展的组织中，可能在团队之间经常变动或者定期的人员流动，同行审查在这种组织下将不足以对关键代码进行审查。最糟糕的情况是：一个初级程序员向另外一个初级程序员展示他或她的代码，第二个程序员说：“你的代码看起来跟我的一样，通过。”没有经验的程序员无法找到很多复杂的错误。尽管如此，当我们看到一个初级审查人向一个非常资深的开发人员解释程序的时候，这将会是一个很有趣的代码审查过程。虽然这可能是一个更耗时的方法，但是培训和开发效果明显。代码审查将会是你的良师益友。

我们建议：关键性的算法、复杂的功能、安全模块和加密模块实现部分应该由拥有正确技巧而且懂得可能引入的漏洞的人来审查。这些审查人员通常是团队里非常资深的人或者是团队里有资格的人。他应该对预期要实现的功能很熟悉；通常，这个人在过去应该成功从事过同样的或者相似的功能开发。

9.2.3 静态分析

静态代码分析是运行一个自动化的工具的过程，它读取源代码或者源代码编译后产生的目标代码。静态代码分析工具通过预处理和错误检测功能来扫描代码正文。因为，正如已经指出的那样，计算机语言的表达能力就是我们所知的计算机科学世界的“重要”问题。静态分析，不仅要了解语言的合法性和语义（跟编译器的工作一样），还必须了解典型的结构和这些结构下产生的错误与误用。此外，静态分析必须创建一个包含所有可能交互的图，以便于囊括有漏洞的交互。用一个有限的、简化的方式来描述现代代码静态分析技术就是：编译器可以发现代码里哪些是非法的，或者哪些是被禁用的。建立在合法性规则上的静态分析器能

发现哪些是“不建议的”。再者，如果这个功能捆绑在一个工业级静态分析器上，它就过于简单了。

相对于 SDL，静态分析如何进行并不特别重要。更重要的是静态分析发现的各种错误，而这些错误会被忽略。对于编程语言来说，直接内存操作、分配、回收都是必需的，已经证明静态分析非常善于识别出代码中内存容易误用的地方。例如，当复制一段数据到基于栈的缓冲区之前不检查副本，或者没有把副本的长度限制在缓冲区长度之内，C/C++ 中常见的栈溢出问题会发生。这是一个典型的安全漏洞。如果外部输入可以进入缓冲区（即通过用户输入的方式），栈溢出漏洞可以用来执行攻击者输入的代码。

大多数静态分析工具很容易通过检测错误的内存操作中的副本的长度来确定错误。一些工具甚至能通过传递给副本的输入方式来确定严重程度。这特别适用于在错误分析^①前构建执行流图的静态分析方法。或者相反，这种分析方法可能会因为没有发现用户输入途径而低估问题的严重性。在后一种情况下，动态分析永远不会发现这个问题，因为没有输入拷贝到缓冲区。在前一种情况下，动态分析可能会也可能不会发现这个问题，这取决于用户尝试的输入。

由于静态分析器可以查看所有的代码，不仅仅是处理输入数据的那部分代码，因此我们认为静态分析处于接近开发人员的程度。一些工业级的静态分析程序能构建一个贯穿全部代码的代码图。这些分析工具建立一个整体视图提供给分析人员，这个视图不仅贯穿了整个代码，而且包含模块间的关系、API 使用情况、数据暴露和隐藏，以及其他微妙的编程模式。有了这个代码分析图，当发现一个错误的时候，分析者可以通过代码之间的关系所提供的更多信息确定问题的严重性。当然，由于静态分析器分析源代码，因此它可以精确地指出潜在的错误所在的代码行。这节约了开发者修复缺陷所需要的大量时间。

除了可执行视图之外，整体的缺点是可能有更少的潜在错误被报告出来，并且不等同于让这些错误完全暴露。事实上，那些未暴露的错误因为运行时的一些限制，当它们暴露出来的时候，危害不是特别显著。也就是说，并不是所有潜在的缓冲区溢出漏洞有同样的危害。典型的例子是缓冲区溢出，它需要非常高的特权才能得以利用。已经拿到所需权限的攻击者没有必要进行一个额外的攻击，来达到运行攻击者任意代码的目的。在大多数现代操作系统中，在这种特权级别下，攻击者可以执行他想要执行的任何代码而不需要额外地利用其他漏洞。换句话说，这样的缓冲器溢出，没有潜在的攻击价值。这种溢出攻击完全是理论上的。不过，因为静态分析器不执行程序，并且没有用户权限的概念，所以分析器不能把高权限的任何代码执行和低权限的任意代码执行区分开。这将需要开发者的分析来证明如下这种报告的缺陷：

充裕的待查项所带来的困窘，在于现代商业静态分析工具一般能够找出更多错误，而单凭用户的才智或者意愿，则是无法与之相比的。^[9]

解决充裕的待查项所带来的困窘的一个办法是开始的时候就采用一个容易理解、高可信度的分析方式。“高可信度”是指：“只报告那些具有极高可信度而且事实上真的存在且必须

① 调用图的优点是，通过代码可以检查每一种可能的路径。调用图的缺点是，许多可能的路径中的枚举可能永远不会被执行。静态分析有有限的途径，以确定哪些路径是至关重要的，哪些是实际上暴露在攻击面上的。分析器采用“猜测”方法。

修复的缺陷。”这意味着你把“攻击性”或类似配置调低到一个低的设定。这些配置只检测到静态分析器厂商所相信的 90% 或者更确信的结果，也就是说，有 10% 或更少的误报。显然，这意味着某些缺陷会错过。俗话说：在攻击面前，任何缺陷都意味着攻击者的胜利。很难实现 100% 的缺陷检测，甚至前几个版本 80% 都有些困难。在一个程序的早期开发阶段减少攻击面或者漏洞到 20% 以下对于构建安全程序（一个 SDL 程序）是很重要的。

推行用静态分析来查找 bug，行政上的坚持有时是一种保障，有时则不过起着拖后腿的作用，但有时又涉及更深层的因素以及文化问题：要避免各种能够被静态分析发现的 bug，主要取决于纪律性，有时这一点在程序员中间并不受欢迎。修复 bug 并查验团队现有的工作成果，是需要适应力和判断力的。尝试设计一套简单的规则和指标来规范团队，在我看来，客气地讲，是不成熟的做法，恐怕还是不可能的。^[10]

从较低的目标开始，来达到高可信度的结果，这样将被更快地接受，因为工程团队能迅速从静态分析工具的结果获得信心。我们建议工程师应该像相信他们的编译工具一样相信静态分析器。从小的目标开始并关注高可信度的目标可能会获得这样的信任。一旦获得信任，团队就可以开始尝试更多的检查，更广泛的缺陷分析和更积极的扫描。不过，根据我们的经验，工程团队信心和信任的关键在于静态分析程序的成功。

换句话说，第一次使用的时候，简单地使用默认或者一应俱全的分析方式是典型的错误。团队将受到难以控制的缺陷总数的轰炸（有时一次静态分析会有成千上万条结果）。缺陷跟踪系统将失效。这些轰炸可能会阻碍静态分析工具的使用。工程师将会对这个可以提供有用和可靠结果的工具失去信心。

相反，建议开始的时候，使用规模小的、简洁的和可控的分析。在这些分析成功的基础上，成功地构建模板、配置和测试套件，这样团队可以轻松、快速地采用它们。让成功的团队协助那些刚起步并且可能举步维艰的团队。让那些减少了缺陷的团队给那些刚刚起步做静态分析的团队布道。我们已经在不同的组织多次成功地使用这种方法。

一个关键是将静态分析放在正确的开发位置。像编译器一样，源代码必须成功通过编译器才能产生目标代码，源代码在构建前必须通过静态分析，没有被不正确地使用，并且不存在已公布的漏洞（如：链接到一个库、可执行文件或其他完全链接和可执行的对象）。静态分析可以被认为是签入和构建过程的一部分；静态分析是鉴定代码是否可构建过程中强制性的一步。

如果分析工具支持开发人员在开发过程中使用，那么工程师们就可以对他们写的代码进行额外检查。程序员将会学习和信任这个工具。他们还将学习安全知识和安全编码；代码安全问题会在编码期间指出。不过，我们相信，虽然在编码期间这么做能带来明显的好处，但是代码在构建到产品或发布前仍然应该再次进行静态分析。这么做可以让程序员有一个优势。让他们尽可能多地交付正确的代码。在一些关键点验证代码的正确性，保证代码必须通过。如果代码是清晰的，那么通过静态分析不会发现多少问题，商业静态分析器一旦配置和调整正确，就可以很快地分析一遍。然而，是否有漏洞才是最重要的。这些铁定不能通过检查的代码可能变成可执行目标文件，然后有可能进入生产使用中。信任这些代码，但要仔细检查。建议让程序员在完全提交并可能发布代码之前，检查自己工作的同时验证实现部分。

在每一个开发生命周期的中心，为了提交源代码，有三个相辅相成的过程。开发人员需

要进行培训，在规定下写出正确的、无差错的、无漏洞的代码。这种做法和规定要求培训、实践和空间，其间可以让开发人员犯错，尝试新事物，学习，并在学习过程中相对比较自由。这需要精锐的管理，来管理那些能干的、有创造力的人，这些人在实践的同时练就他们的技艺。编写代码后，安全构建过程中的下一个核心步骤是人工代码审查。像安全编程一样，人工代码审查不仅是规定，而且是实践。人工代码审查也是学习和掌握代码的机会；人工代码审查的另一个作用是编写正确的代码。出色的代码审查员会成为出色的程序员。出色的程序员会成为出色的代码审查员。这两个过程一起携手，提供更清晰的代码和更有经验的编程人员。最后，开发过程中的中心地带是静态分析，它检查和补充代码审查。静态分析与人工审查相辅相成，静态分析是一种自动化过程，在整体上看它在同一时间跨了多个维度。不管是瀑布模型，还是敏捷模型，安全编码、代码审查和静态分析应该在任何安全软件开发声明周期中处于中心地位。

9.3 确定每个项目的正确行为

七个确定性问题

我们所工作过的每一个组织经常反复地、不断地从开发人员和工程师团队那里发现，增加安全性将是交付产品的另一个障碍。需要注意的是，大多数软件企业都奖励至少在最短时间和预算内交付软件功能的行为。通常情况下，当一个安全软件开发的实践开始或者改善的时候，及时交付的焦点就会被嵌入到该过程的时间点中。开发团队自然会想知道，在他们已经很繁忙的时间表，他们如何成功完成一系列的额外任务。

显然，某个人可以认为，开发人员应该已经将安全集成到开发实践中去。有趣的是，很多时候，开发团队会回应这个事实上，他们确实这样做了：生产“安全”软件。在这里，“安全”是一个有效的术语，“安全”没有精确的含义，其含义太丰富了。

深挖一点，我们就知道软件的验证机制在构建和测试期间已经被仔细思考过了。或者，也许，在网络上的所有通信可以放置在一个加密通道（通常是，TLS）中。极少情况下，在实例化一个强大的 SDL 程序之前，需要做团队整体回应，认识到任务的范围必须得到关注，从设计到测试。这正是使用 SDL 所要解决的问题：整体，内置安全，面面俱到，从头到尾，端到端。

因为安全的任务通常表示为一个任务矩阵，所以团队可能会看到大量的任务，从而变得不知所措。一而再再而三地，我们听见团队领导和开发经理问：“我要做什么？”这个问题的答案真的应该取决于手里项目的类型，安全架构师可能会回答：“嗯，那要看情况了。”这个答案不会让那些希望能够及时有序交付软件的人满意，像产品经理、项目经理、技术牛人、开发经理。

然后，安全人员会问：“我的项目的最低要求是什么？”找一个最小的任务集肯定是相关的而且和有价值的问题。答案是：“嗯，这要看情况而定。”再次不满意。我们意识到安全架构师作为唯一的仲裁者，他所必须做的事常常使整个开发过程变得很神秘。计划变得更加困难。再次，安全被视为是阻碍和障碍，安全被视为不是一个必需的可交付成果，不必作为生产软件的一部分。

经过多年来我们在不同的开发小组内采用多种企业内部组织内所收集的大量试验和错误，

我们固定了项目经理、架构师、技术和工程领导以及产品经理很容易回答的一套问题。这些人通常能理解评估计划的架构和设计的变化，例如：是否要有增加敏感数据和第三方代码，接口类型的添加或更改，以及预期的部署模型。这些维度中每一个的变化都导致影响安全活动必须执行，以生成一组正确的安全特性和需求。其中的一些维度确定需要什么类型的安全测试。合起来，这些问题的答案将所需的 SDL 安全任务流映射到单个项目环境中。

我们不推荐一个“全做”的方法。在没有彻底分析安全架构、没有提供额外的安全价值、没有任何实质性的改变之前增加威胁建模。另外，需要没有价值时不会产生信任的安全的判断过程的这一步。工程师们经常会发现没有价值的活动（他们往往是聪明的、有创造力的人）。没有人喜欢在无用的管理机构浪费时间。大多数工程师很快就会意识到当没有 Web 服务器的时候，动态 Web 测试没有价值。相反，让团队锻炼自己的技能，当合适的时候，让他们回答一些基本问题，合理地添加活动。

这些问题可以问在前面，在开发生命周期的开始，或者在适当的阶段。时间是至关重要的。回答 SDL 中每个错过了恰当时间的相关联活动问题会导致延迟，所需要的安全任务将被错过。只要在需要结果之前回答每个问题，安全结果将是相似的。体系结构问题可以在任何架构开始之前处理，设计问题在设计之前处理，测试结果需要聚焦于测试计划，等等，这些都贯穿了整个生命周期。然而，在一开始确定那些负责的预算和资源分配之前，需要回答这七个问题，以收集需要在开发过程中要完成哪些任务的重要信息。

1. 提出的变化是什么？（以下答案是互相排斥的；选择只有一个。）

a. 架构将是全新的或者是一个重大的重新设计。

b. 架构有望改变。

c. 安全特性或功能将添加到设计中。

d. 如果必要，既不改变架构也不增加了安全功能（即，以上都不是）。

2. 会添加一些第三方软件吗？是或否

3. 会添加一些客户数据（个人识别信息 [PII]）？是或否

4. 这个组织或其合作伙伴操作过其他系统吗？是或否

5. 是否包含 Web 服务器？是或否

6. 程序会有其他的输入吗？（即，非 Web 输入、配置文件、网络监听器、命令行接口，等等）。是或否

7. 这是一个主要版本吗？

尽管是在这个过程的早期，但是这一概念已开始成形，重要的问题是：“有什么是新的？”也就是说，在这个项目里，通过努力，能找出多少改变。这个问题应在架构层面问，有四个可能的答案：

1. 一切都是新的。这是一个新建项目或重大的重新设计。

2. 体系架构将发生巨大的转变。

3. 安全特性将添加到体系结构或设计中。

4. 以上都不是。

提出什么修改意见？

当一切如新时，有一些预先架构的活动，可以帮助确定该组的需求和特点，这些活动将满足该软件的使用目的和部署的安全挑战。图 9-4 显示了全新的项目任务流，涉及重大的重

新设计，还有在系统中从未出现过的安全审查。到目前为止，拥有一套完整的需求，已经被证明能够提供更具有包容性和有效性的架构。完整的需求必须是安全的。目标是“内置安全”不要事后处理。如果所需的功能不包括在体系结构要求之内，它们很有可能不会建立。这迫使部署团队用临时的、不标准的实现来构建涵盖运行系统或者基础设置的保护措施来弥补缺失的安全特性。

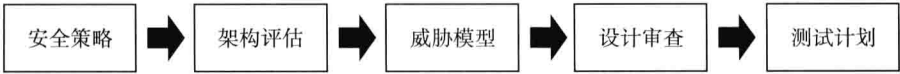


图 9-4 当新建或重新设计项目时的架构任务流

当所有或几乎一切都是新的并且没有现成的时，架构师必须考虑一下遗留体系结构或设计。这有机会详细考虑当前的威胁和它们的攻击，但软件也必须防止自己受到未来的可能攻击。在早期，这种软件重用思想在世界上是罕见的。把时间和安全系统所需作为一个整体具有一个很大的优势。软件安全策略最好在做出开发决定前完成，否则这将会使安全代码设计完全被排除在外或者很难实现。

如果架构是变化的，那么该过程在开始就应该进行架构评估和威胁建模。图 9-5 描述了任务流。这些变化必须鉴于现有的体系结构进行检查，以检查出任何额外的安全要求。这里的假设是，对现有的架构进行了评估，并建立了威胁模型，以确保适当的安全需求被内置在设计之中。在这种情况下，如果从未有过架构评估及威胁模型，那么体系结构应被视为新建项目。

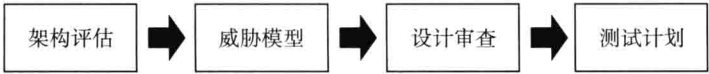


图 9-5 当架构改变时的架构任务流

即使没有增加或改变现有的架构，增加了存在安全隐患的任何特性都表明有必要重新进行设计工作。该设计使得架构可重建。程序员根据设计工作。因此，重要的是，任何安全需求或功能设计必须完全正确。安全专业知识是至关重要的；设计审查的目的是为了确保适当的安全专业知识运用到设计中。图 9-6 显示了这两个设计相关的 SDL 任务流。

像任何其他特性或功能一样，在测试计划中，每一个安全功能必须进行彻底的正确性测试。创建测试计划是设计工作的一部分。测试计划是设计的神器。

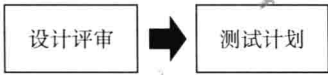


图 9-6 当设计安全特征时的设计任务流

“什么是新的？”这个问题的答案是一个列表性的选择。有且只有一个选择。这些问题的答案不是互斥的。相反，第二个选择，“架构的变化”，是第一个的子集；“新建项目”意味着该体系结构将发生变化。答案，“安全特性”，独立于前两者。“安全功能”是为了赶上该项目在设计时必须考虑的变化。如果架构是正在变化的，它可以推定该设计也将改变。

剩下的问题的答案是“是 / 否”，二选一。

任何第三方软件会增加吗？

添加第三方软件产生了两个挑战。

1. 额外的软件可能有安全漏洞，然后成为最终成品的一部分。一般来说，对于商业性质的软件，开发团队没有软件的源代码。一个组织将完全依赖于第三方来修复漏洞。或者，如果软件是开源的，开发团队选择修复安全缺陷时可能会有法律和资金问题。根据许多许可证，修复缺陷后必须在开源社区回馈给大家使用。这在商业上可能是不可行的。

2. 是否修复第三方软件的缺陷是关系到要添加的第三方软件的许可证问题，这是一个难题。要包含这些功能，不同的开源和免费许可证在一个商业组织什么能做、什么不能做的限制方面差距很大。你售卖产品的能力可能会受到限制。或者专用知识产权可能不得不暴露出来。基于这些考虑，因此我们认为在使用第三方软件之前，咨询熟悉软件许可证的专家（通常是法律部门的人）是必需的。软件许可专家将彻底审查许可证的细节。

肯定地回答这个问题对于所需的任务增加了一个法律许可审查。它可能还需要需要监控第三方软件的安全补丁（只要包含第三方软件）。

第三方 SDL 的组任务，以及这些任务的线性流，显示在图 9-7。



图 9-7 第三方和开源任务流

任何客户数据（个人身份信息）会增加吗？

如果你的软件将面临隐私挑战和法律法规，就必须确定是否个人身份信息将由正在开发的应用程序处理。根据组织的不同，软件可能要负责客户信息的安全，并满足法律规定的隐私规范。可能存在一个专门的团队致力于隐私保护，或者作为法律的一部分，或者作为其他组织的一部分。无论团队向谁负责，这些都需要数据保护审查。因此这个问题，没有专业隐私方面的知识，“你添加客户数据吗？”不是一个容易回答的问题。

我们回答这个问题的措辞是“添加了客户数据”，因为，根据不同的定义，个人身份信息的定义差别也很大。这些组织受到多个司法法规的管辖，必须确定它们将如何满足事实上法规可能存在的冲突。一般情况下，做出这些决定需要隐私方面的法律知识。大多数开发团队没有这方面的法律知识，无法理解各种各样关于个人身份信息的法律法规，无法在程序中遵守各种规定。然而，我们发现，开发团队很容易了解应用程序是否处理了客户数据。根据开发团队的专业知识，你可能想让适当的人选或隐私专家决定是否有关隐私问题要回答。简单地标记客户数据，至少保证团队正确地处理隐私问题。

精心设计和精心实施数据安全保护是信息安全的目的之一。然而，应该指出的是，隐私不是客户数据安全性的唯一标准（和法律标准）。隐私也可能包括政策、展示、同意，限制地域运输和储存数据，以及其他与安全设计和实现相关的标准。因此，安全性不等于隐私。另一方面，能够遵守一些隐私法和期望，当然需要足够的安全。而且，当然，数字安全的许多方面都是独立的私密领域。

该组织或者任何合作活动会主持任何系统吗？

系统的部署模型和执行环境不仅影响需要的安全特性和控制，而且在更高层次的抽象中，

这些安全保护层的控制和特色将会被过滤掉。不是每一个系统都需要所有的控制。那么如何选择呢？

软件用来由生产商以外的其他人进行部署，例如，由客户来部署，他必须有足够的能力从客户的实际环境来部署程序，以满足客户的安全状况。安全的目标是授权给客户，同时不降低客户环境的安全状况。

另一方面，软件将在可感知的边界中托管，这些边界承载一个组织中根本不同的安全责任。我们使用术语“可感知边界”，因为客户、媒体和公众影响力往往不能区分“合作伙伴”和组织之间的不同。在一次成功的攻击面前，最大的或最知名的实体或品牌将承担失败的责任，无论到底与失败之间有没有关系。鉴于这方面的认知，我们建议采取负责任的开阔视野。当然，每个组织必须确定在其可接受的范围为事件负责。

系统由一个已知的和可控的基础设施托管，并继承了基础设施的安全特征。因此，系统将会被那些假设系统不可能部署在组织外部的组织部署，这在组织边界之外，在第三方的前提下。可以假设在哪些安全功能将内置于待部署系统上的基础设施中。的确，不好的消息是，本地部署的系统在一个已知的基础设施下会存在一些缺陷。也就是说，组织上托管的系统继承了其所部署在的基础设施上的安全状况。图 9-8 表示一组相关的任务部署到一个托管的基础设施上。

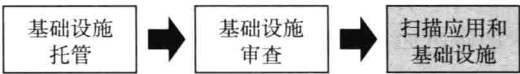


图 9-8 托管系统的任务流

托管系统还必须满足所有当地的政策和标准。这使得配置、硬化和调优更具体。因为软件要被其他人部署，所以设计成能应对未知的安全需求就显得很重要。在托管软件的情况下，这个需求通常是很明显的；这些需求是可以预想、设计和预构建的。

为了理解安全需要准备什么以及需要减少什么缺陷，在需求收集和设计架构期间重要的是要问：“组织上托管？”。如果那些负责安全需求的人（通常是安全架构师）不熟悉该系统将被部署在的基础设施，那么必须对基础设施进行评估。同样，为了将它们应用于该系统，策略和标准必须彻底理解。

通常，对于一个正在运行的安全架构实践，在现有基础设施上会有一些专家和政策专家。对于托管系统，这些主题专家可以帮助细化分析，以精确地满足部署基础设置用的系统的需求。通过这种方式，托管系统将能满足组织的政策。另一方面，如果它是一个新的基础设施，有一个尽职调查的责任，以检查每个安全因素是适用的。坚持把在新的基础设施上进行研究作为系统分析的一部分。

如果不看基础设施，而是仅仅着眼于正在建造的系统，那么尽管架构师努力建立一个良好的纵深防御系统，攻击者仍然可以通过基础设置系统进行攻击。构建安全体系架构必须有责任尽职地调查，始终是从前到后，一边到另一边，从下到上，深入分析所有支持或交互的组件。连接的每个系统会打开不同的可能性，即互联系统的安全状况将以某种方式会影响到它所连接的系统的安全状况。这是安全架构分析的基本定律。安全分析失败了就是彻底的失败。如果基础设置在组织的政策控制之下，该分析必须包括将要部署系统的基础设置。因此，总是要问：“该系统的任何部分都由这个组织或其合作伙伴托管吗？”

包括一个 Web 服务器吗？

正如这本书所写，Web 服务器安全性测试需要使用一套专业的工具和技巧。出于这个原因，我们发现确定是否需要 Web 服务器测试是有用的，这样可以使相应的测试参与进去。一般来说，软件中的每一个输入必须经过彻底的测试。专用工具可以测试自定义 Web 应用程序代码。这些工具需要专业知识，不仅理解网络攻击及其变种，而且每个特定工具有它的长处还有它的局限性。所以我们在整个测试计划中包括一个单独的方法，来确定 Web 服务器的存在性，使得 Web 输入需要特殊关注。这种类型的测试称为动态测试。虽然各种动态测试工具中大多数一定有重叠，但是我们还是建议使用针对 Web 应用程序的这些工具。重要的是，许多攻击者专注于 Web 漏洞。当然，如果系统暴露在公共互联网下，它将会不断被攻击，必须让自己做好抵御这些常见的变体攻击的准备。然而，即使系统被部署在一个更可信的网络上，我们相信，高强度的 Web 服务器动态测试仍然是很重要的。高明的攻击者有时会在可信的网络上展开攻击。防御薄弱的 Web 服务器本身就是一个目标，或者作为一个跳板，攻击其他更有价值的目标。因此，良好的实践将使所有 Web 服务器从这个专业测试受益。

该程序会有其他输入码？

除了 Web 服务器输入以外，还有无数其他输入可能内置在系统中：命令行接口、配置文件、网络输入、原生形式、脚本引擎，等等。这些是能被攻击的地方。每个输入有可能存在一个潜在的逻辑错误，它能绕过如身份验证和授权这种访问控制。程序的逻辑错误可能被滥用以做意想不到的事情，能降价销售或使用不存在的价格销售产品。此外，如果程序员使用编程语言直接操纵计算机内存，不完善的输入处理可能造成程序或者操作系统的严重滥用。与内存相关的错误通常允许系统提权或允许执行攻击者选择的代码。输入在任何程序中都是一个重要的攻击手段，经常是主要的攻击面。

写测试计划之前，所有输入到系统的数据应该被枚举。每个枚举输入应该被测试，以确定正确的输入值被正确地处理。此外，必须运行测试以证明不正确的输入变量不会导致系统崩溃，或者更糟糕的是，越过软件边界作为一个攻击向量（例如在缓冲区溢出允许提权和攻击者执行的代码中）。常见的工具是编写一个模糊测试工具。这些工具模拟许多变量，攻击者可能对软件进行攻击。一旦模糊工具准备好一个特定的输入点，它自动产生一组变量并尝试这些不正确的输入。当程序开始表现异常或崩溃的时候，模糊工具将停止检测。这样，不正确的输入处理，甚至不正确的内存处理，将会暴露出来。建议在整个开发周期内变化的每一个输入都被模糊测试。

模糊工具和 Web 漏洞分析工具之间有一些功能重叠的地方。如果测试时间不长，我们建议对 Web 服务器运行 Web 漏洞分析工具，对剩下的其他输入进行模糊测试。然而，Web 漏洞分析工具和 fuzz 工具重复扫描漏洞并不是一个坏主意。因为 Web 扫描工具寻找已知的攻击模式，模糊工具检查没有处理好的输入，采用这两种类型的动态分析比单独使用其中一个提供了更好的覆盖范围。

测试计划将可能会包括预期的安全功能和动态测试两种测试，通过人工审查和静态分析，以确保没有漏洞，如图 9-9 所示。



图 9-9 测试计划的组成

这是一个主要发布（版本）吗？

第 7 个和最后一个问题是关于这个项目的重要性或发布软件作为一个整体的关系。在软件开发生命周期将增加的部分被视为一个重大补充和 / 或修改？如果有大量的体系架构更改，可能伴随着在当前正在运行的代码或维护版本中增加了显著的特点，可能引入微妙的逻辑变化，甚至可能有错误进入代码库。一些最微妙的 bug 会对其他的代码片段产生意想不到的负面影响。事实上，很难对内部交互行为进行彻底检测；当添加的有负面影响的新功能与遗留代码以无法预料的方式进行交互时，尤其如此。

此外，软件的使用者往往对较大的修改有更高的期望。主要版本往往将会比小幅更新版本获得更多的审核和更广泛的发布。

由于这些原因，当一个主要发布版本被认为“代码完成”的时候，很值得对其进行全面、整体的安全属性和漏洞检查。可以通过一个独立的攻击和渗透来进行检查。“独立”在这种情况下意味着独立于构建软件并测试它。由于渗透测试需要大量的技巧，因此可以聘请专家。我们还应该记住，每个测试人员提供的结果是不同的；每个渗透测试对于测试人员和被测试软件来说或多或少都是独一无二的。因此，在开发模型、执行栈上有重要技能的人，甚至是编写这个软件的人，执行这个主要发布版本的渗透测试将是一个良好的实践。如果不匹配测试软件类型可能会带来不理想的结果。你想找的测试人员应该知道如何攻击你写的软件。

通过高度熟练的攻击者对软件的攻击，你可以发现精明的攻击者可能会发现并利用的各种各样的漏洞。微妙和复杂的错误可以在软件正式上线前被发现和修复。此外，你将对这个版本中任何残留的风险有个更清晰的认识。

这样一来，你可以向组织和消费者保证软件确实不但是自我保护的而且是正确的。我们发现，市场和 / 或广泛发布的软件竞争都很激烈；经过渗透测试证明可以具有竞争优势，这表明该软件不仅是安全的，而且是软件制造商对客户安全的承诺。

回答这 7 个确定问题可以控制瀑布和敏捷流程的安全任务流。在每个流的开发不同处放置适当的活动。这些差异将在下面描述。

这 7 个问题中每一个的答案回答了“什么是我的项目必须执行最小的一组活动？”然而，不应向团队这些答案，如果他们不能对最小设定补充其他活动，而这些活动可能会提供附加的安全值。举例来说，如果有时间，甚至现有架构和设计的威胁建模可能会发现一些有用的东西。攻击模式随时间而改变；昨日流行的攻击方法可能被另一种更容易或成功率更高的攻击方法取代。

问：“什么是新的？”来评估建议的变更范围。有迹象表明，每个答案流向其他活动。而在这些迹象的早期，更多的架构活动将会引导后来的设计、构建和测试任务。表 9-1 和表 9-2 描述活动的选择。

表 9-1 SDL 确定问题所指示的架构和设计活动

问题	步骤 1	步骤 2
未开发区或者重新设计?	确定所需的安全状况和包含利益相关者的安全战略。生成高级安全需求	执行“架构改变”步骤流(见图 9-4)
改变架构?	审查安全架构并建立架构威胁模型。得出架构需求	执行“添加安全特征”流(见图 9-5)
将要添加任意一个安全特征?	设计一定要审查以确保正确性和评估逻辑漏洞	对每个安全特征添加功能测试到测试计划里
是否添加客户数据(个人可身份信息 [PII])?	执行一个私有审查	
是否一些系统会被组织或者合作伙伴托管?	审查基础设施安全性	在基础设施里的软件被发布之前执行一个第三方库漏洞测试

表 9-2 SDL 确定问题中的代码规范和测试活动

问题	步骤 1	步骤 2
是否添加了第三方库软件?	执行一个法律许可证审查	软体发布后之验证与测试, 监控在第三方软件中发现的漏洞
是否有一个瀑布构建周期或者敏捷冲刺周期?	安全编码。执行手工代码审查。运行静态分析	修复发现的缺陷
是否有一个 Web 服务器?	执行 Web 动态漏洞测试	修复缺陷
是否需要枚举其他输入?	对所有非 Web 输入执行模糊测试	修复缺陷
是否有一个主发布版本?	考虑一个独立的漏洞评估和渗透测试	

表 9-1 列出了适用的活动, 无论你使用的是瀑布模型还是敏捷方法。这些都是以前描述的几个部分。

对于一个瀑布模型开发方法, 下面的任务流以线性方式进行: 设计, 构造, 然后进行测试。对于敏捷开发方法, 以下任务将在每个开发周期(并列“冲刺”)迭代。而不是单个设计时期必须在开发开始之前完成, 每个短周期被设计为已选定版本的增量。该循环将完成时, 这些功能已经过测试, 并准备发布, 即“代码完成”。因此, 在敏捷开发中, 有短期、集中的设计周期, 并进行编码, 然后再测试。然而, 据我们所知, 对于任何一个特定的增量进行重新设计的情况并不多见。无论何时, 有代码需要进行测试, 测试都有可能开始。出于这个原因, 该表包含每个开发周期的安全任务, 在敏捷开发中根据需要通过敏捷团队来执行。在一个敏捷开发中所有三项活动可能同时发生。表 9-2 显示了一些活动, 这些活动属于“构建”瀑布模型或敏捷过程及其相关的确定问题的一部分。

9.3 节将全面研究这些任务流。

重要的是要评估那么没有经过正式 SDL 过程的遗留代码和项目的安全影响。当早期的开发没有使用一个成熟的 SDL 过程(也就是在当前的变革之前)时, 就需要尽职调查, 以决定是否必要评估前期工作。在一个全新 SDL 过程的情况下, 可能会认为代码或项目以前没有进行过审查; 询问而不是假设, 始终是一个很好的做法。确定问题的措辞是, “要补充……吗?” 因为这一过程假设在先前的开发周期中安全一直按照 SDL 进行。因此, 7 个问题集中在什么已在发生改变, 而不是任何继承的遗产。在实施或更改 SDL 时, 任务之一是当应用新的 SDL 到当前的开发中时, 评估你的遗留代码和项目将需要多少变化。你的安全技术债务是

什么？存在多少设计失误？在你当前的开发周期引入了多少有漏洞的代码？

解决 SDL 遗留技术债务的一个方法是问，“我的应用程序改变得有多快？”在快速变化的时期，也许这些债务直到被偿还前还是有意义的。

与此同时，在遗留代码传承到可预见的未来的情况下，削弱漏洞和风险也许是有意义的。一个成功的方法是致力于每个月找一天“打击 bug”。开发人员休息一天不编写代码，而是在修复旧的缺陷。这样的“打击”可以作为一种放松，从严格的代码编写中解脱出来。这样一来，“技术债务”减少了，并且缺陷从产品代码里一点点被删除。

bug 打击一般不针对设计问题。通过架构和设计特性来提高现有产品的安全性必须考虑对其他特性的要求。如果销售的产品正在由于缺乏安全特性被错过，那么这些特征的价值应该是显而易见的。然而，通常客户的安全人员与软件生产商的安全人员在一个单独的对话内交流。安全人员能相互理解。此外，客户的安全人员想要获得一些保证，保证供应商的安全人员在软件设计和编写期间都参与了软件的安全性建设。产品管理必须包含在这些客户的安全对话里。产品经理需要明白，客户的安全团队往往会投否决票，产品可能为了解决例外情况来部署软件而花费额外的资源。当产品经理和安全架构师意见一致的时候，安全特性可以在其应有的位置为客户增强体验，而不是作为一组不可行的需求存在。

任何安全需求都来自于评估遗留软件需要被添加到积压的特性要求中。这些应优先通过风险评级。风险评级可以考虑客户的需求。有可能的话，一个安全主题专家应该包括在这个对话里。

最后，解决技术债务是一个具有风险的决策。如果解决比损失更昂贵，解决的风险可能是没有意义的。各种因素必须权衡，包括没有发现新技术时丢失的机遇成本。我们已经看到大型代码库已经被列出的多达 75000 项缺陷的情形。一组已知的自动化工具在撰写本章时还不够成熟，因而无法保证每一个发现的问题实际上确实是一个缺陷。简单地认定其中 75000 项缺陷实际上确实是缺陷是显而易见的琐事。更不用说修复名副其实的缺陷，我们应慎重考虑这件事。重要的是记住，Bran Arkin, Adobe 的 CSO，告诉我们其中一位作者，“漏洞不一定都可利用。”事实上，漏洞不一定是漏洞的。当漏洞被刻意的部署模型暴露的时候，一个基于风险的方法将专注于利用这些漏洞。原始缺陷数量，对于其漏洞本身是没有意义的。

9.4 架构和设计

系统架构是一种常见学科，用于处理对象（现有的或待创建的），这些对象也称为“系统”，在某种程度上它支持推理关于这些对象的结构性质……系统架构是描述的实际困难和复杂系统的设计在概念上的响应。系统架构有助于一致性地描述和有效地设计复杂的系统。^[11]

为什么在 SDL 中包括架构呢？目前信息安全领域有一句名言：“内置安全，不要螺丝。”架构是系统的结构、流程、数据。关于该架构的决策可以从根本上影响系统的安全状况。如果未能添加一个身份验证机制，那么最好在架构设置后加上它。最坏的情况是，有要求认证的需求，但认证机制不可以添加到该架构中：验证机制已经固化下来。

一而再，再而三地，我们看到一些系统，假定它们运行在一个高度受限的网络中。假设网络将提供适当的限制，然后设计应用程序在这个架构下不采取任何保护措施地在组件之间

传输敏感信息，其实数据流可能不被保护，因为只假设目标系统中的组件部署在受保护的网络上。在异构网络、高级持续性威胁攻击和巨大的云服务器场环境下，任何应用程序都把自己限制在严格受保护网络中的可能性非常之小。绝大多数的网络是共享的，很少有高度可信的网络部署。假设网络将保护系统的所有组件和组件之间的所有数据流将是一个重大的架构错误。然而，我们不断地看到这种架构。我们需要在共享网络上暴露组件、数据流和在共享环境下管理复杂的防火墙规则之间做出选择。此外，防火墙可能无法提供用户所需要的深度应用防护功能。

安全架构具有特定的特征。

- 安全架构都有自己的方法。这些方法可能是一个离散的安全方法的基础。
- 安全架构有自己的离散的看法和观点。
- 安全架构提出的非规范化流经过系统和应用程序。
- 安全架构经过系统和应用程序引入自己的规范流。
- 安全架构引入独特、专用组件的设计。
- 安全结构要求 IT 架构师具有独特的技能。^[12]

绝大多数的架构设计选择至少会有一些安全隐患。其中一些选择将具有深远的安全影响，这些影响表明安全控制中的特定模式。因此，有安全格言，“内置安全。”构建安全如图 9-10 所示

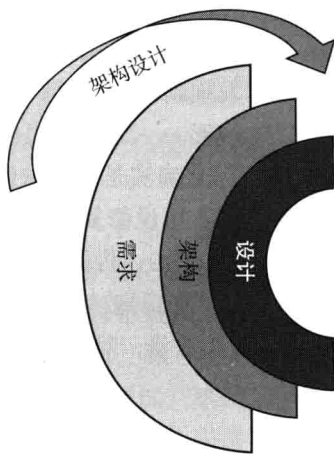


图 9-10 需求策略：架构，威胁模型，设计审查

通常，架构流始于需求收集。源自需求，满足需求，然后提出了一个架构细化迭代。这适用于一个全新的系统以及改变一个现有系统。当然，其他影响因素包括哪些目前可以构建，哪些目前存在，哪些是一般架构上未来的目标和策略。当前的能力对提出解决方案产生深远的影响。这才是真正的安全性，其能力丝毫不亚于数据库、网络、服务器类型，以及支持这些的人员和流程。事实上，最大限度地提高现有能力可能是要求之一。使用现有的验证机制，可能是该系统的安全要求之一。

我们强烈建议安全性包括在需求收集阶段。正如我们所指出的，内置的安全功能将会是整体系统安全蓝图的一部分。一个 Web 系统如果可以访问财务信息，该系统将有可能需要进行身份验证以确保该信息仅提供给信息的持有人。这样的系统也可能会有一个授权机制，以

便给予合适的用户正确的访问。事实上，这样的 Web 系统也将具有其他不太明显的安全性要求：固化系统，这样他们就可以抵抗不可信网络无处不在的攻击；小心地分层，让前端系统甚至中间系统都不能直接访问到数据；从业务逻辑到数据存储的验证，让来自不可信或者不恰当应用的不恰当访问得不到授权。这些需求是显而易见的。然而，我们已经看到许多系统忽视了收集特定的安全需求的必要性，忽视了项目成功的风险。我们甚至看到了安全需求表述为：“该系统将是安全的。”很显然，这些非特异性的、广义的要求没有用。

企业安全架构是由整个企业架构的组件专门设计来实现……总体目标……保持一个组织的信息的可用性、完整性和机密性。^[13]

对于一个全新（或完全重新设计）的架构，安全性应及早考虑，以从战略上确定如何能够满足系统未来以及目前的预期、改变和增长。安全架构师必须在应对目前的各类威胁和这些威胁在类似系统目标上的攻击时具有坚实的基础。在这些战略决策上，安全架构师还应该对于新兴威胁和攻击方法的趋势有一个良好的预感。什么新的威胁才刚刚开始变得活跃？这些新的威胁中，它们可能的攻击方法是什么？当威胁主体的组织和复杂度增长时，它们可能怎么扩大攻击模式？对于这些各种各样的问题，好的架构不仅可以用于现在的用例，而且在可预见的未来也可以使用。通常，企业级架构师考虑类似的问题，比如组织的成长，用户数量的增长，数据的增长，以及架构扩张的能力。未来的安全需求需要跟现在的安全需求一样被给予关注。

任何架构评估要求必须满足评估的架构。通常情况下，早期的要求是比较一般性的：用户将被验证，系统将需要固化，支付卡行业（PCI）认证（在适当的层次）将需要满足，等等。这些细节被合并到新的架构里。

由于架构在系统里被很认真地处理，因此安全需求也将开始采取一些特别之处。选择一个特定的验证系统：对于一个大的服务器场，例如，选择一个系统，它可以每分钟处理数以百万计的验证，能够处理数以百万计用户身份，可以在可接受的时间和环境下完成处理，等等。或者，如果该验证系统是很合适的，或许用一个一体的库或者另一个模块来实现将会是足够的。使用前者意味着巨大的增长和大量的用户流量，甚至异构系统。当使用后一种验证系统时，较小的库可以阻止服务器场数目的增长。在考虑到预期用途（比如，为客户部署的设备验证系统）时，一个相对的制约机制可能是必要的。在任何情况下，特定的选择将基于该系统的在预定部署和相对于预期增长的要求下进行。该架构将变得更加具体和特殊。安全架构过程的输出特定于组件，这些组件提供特定的服务和众所周知的通信协议。

对于现有架构中的系统，这种架构的任何改变可能都会有安全隐患，所以每个架构变化导致的安全问题应予以考虑。举例来说，添加一个第三方的合作伙伴，其财务数据流可能导致要引入额外机制，以在传输过程中保护金融数据。另外，保护需要落实到位，使得只有特定的伙伴将能够交互。换句话说，现有架构的安全需要将会改变新的组件，或者新的通信流，或者新的数据类型。更改任何架构部分必须考虑整体架构和所有现有的安全服务。如果不同于设计工作，这项工作非常相似。因此，在考虑的系统安全策略后或者一个现有系统的架构改变之后，指定系统安全性的架构评估。这组 SDL 任务流假设现有架构对于一个全新的系统已经通过一个全面的、彻底的安全评估。如果没有之前的安全评估，那么现有的架构应该被视为全新的。

一旦架构及其所有辅助组件,包括部署模型和任何基础设施,是可以彻底理解的,就应该建立威胁模型。威胁模型包括设置信任边界和确定攻击面。并不是每一个攻击方法都跟每一个系统都很密切。重要的是:

- 部署模型打开或关闭攻击面。
- 执行环境(继承自底层系统)提供服务 and 攻击面。
- 在现有的基础设施上部署系统的时候遗留有安全漏洞和缺陷。
- 不同的语言和运行时模型提供独特的优点和缺点。

一些组件将会更加暴露,并因此而不可信,而其他组件可能需要被屏蔽,或者停留在信任边界的核心。

假设一个用户界面和一个软件之间的交互必须成为操作系统的一部分(例如,一个内核驱动程序)。用户软件将成为一个明显的攻击点。攻击者希望窃取用户的数据或通信内容。攻击者可能会通过内核驱动程序的用户界面控制操作系统。保护不善的内核驱动程序也可以成为提升权限的一个途径。内核驱动程序是操作系统中最值得信赖的核心的一部分。用户界面组件应该保护自己,内核驱动程序也应该保护自己。然而,需认识到如果用户界面保护失败,那么内核驱动程序必须保护好自己。驱动程序还必须保护操作系统,并且不给内核添加漏洞,保护操作系统的最高特权。

用户界面和内核驱动程序之间的自然信任边界是显而易见的。下面这些应该也是显而易见的,专为简单系统设计的威胁模型不仅考虑用户界面、明显的攻击面,而且考虑用户软件和内核之间的流或交换。用户软件可以选择信任的内核驱动程序。但是,在任何情况下,内核驱动程序接受来自任何模块(除了用户界面)的通信。此外,用户界面必须具有高度的自我保护性以避免试图把它作为获得内核权限的一种方式。从这个非常简单的威胁模型,你可以看到新出现的安全要求必须内置在架构和设计中:

- 用户界面输入验证
- 模块间验证
- 来自用户界面的内核输入验证(在身份验证或者用户界面的输入验证失败的情况下)

根据操作系统的不同而不同,关于加载顺序、执行权限,安装机制、配置文件权限、配置文件验证等可能会产生大量的需求。

我们不相信威胁建模本身就足够了^①。威胁模型需要输出对架构的分析:完全理解所有的组件、流和数据。架构分析确定安全功能的需求(即满足潜在用户的安全需求)。如果没有这些信息,威胁模型可能是不完整的。有了架构结构和安全需求,可以威胁模型建立以了解当前系统面对的攻击面。如前所述,因为每个攻击对于每个系统都是不可靠的,所以威胁模型认为可能的攻击场景基于对相似系统的优先攻击模型的分析。

从威胁模型输出的任何要求,将创建新的测试计划项,这是真实的。这些新的测试用例确保安全需求已正确建立。因此,如果威胁模型产生需求,测试计划将获得新的测试用例。安全测试用例依赖于架构分析和威胁建模。

① 威胁建模系统,如微软的 STRIDE,可以做我们做不了的假设。STRIDE 假设建模者已经相当熟悉目标系统的架构,因为任何成熟、异构的开发团队都可能会很熟悉。然而,对于高度共享的安全架构师,可能并非如此。此外,STRIDE 的目的是授权未经过安全训练的开发团队开始。因此,STRIDE 故意被简化。我们赞赏 STRIDE 和类似的方法。如果你的情况符合 STRIDE 的目的,我们鼓励您使用它。

一般来说,如果没有架构上的改变,那么架构分析和威胁建模可以忽略。(这假设现有架构已经经过安全评估和威胁建模)。

系统的设计必须实现所有来自架构的需求。如果一个架构是结构、流程和数据,那么按照预期的结构实现设计。设计必须有足够的特异性,来保证它实际上可以编码。

给予充分的、明确的、足够详细的需求或者用户故事(Scrum),熟练的软件设计师很容易把架构及其需求转化成软件设计。这可以在瀑布过程的编码阶段之前完成,或者在每个敏捷开发的增量生成周期之前完成。在这两种过程的情况下,特别注意安全特性和需求是非常重要的。这些都必须绝对是绝对正确的,否则实现可能产生了漏洞,或者更糟的是,创建一个新的、不受保护的受攻击面。逻辑已经得到了无懈可击的关键安全功能,如加密例程或授权方案。该系统的用户将根据这些功能来保护自己的资源。

我们把安全设计审查放在这么一个点上,设计师认为这个设计是非常接近完成的。安全设计审查应由理解架构和功能的安全审查员执行;通常,审查最好由有设计经验的,甚至实现过安全特性的人来做。我们进一步表明,审查员是独立于设计团队的。同行审查是一个强大的工具,可以用来验证设计的正确性和完整性。同行审查的另一只功能是常常可以发现那些设计中留下的可能已经疏漏的错误。如果任何安全特性或需求都建立在当前周期上,就执行安全审查。

正如已经指出的那样,计划设计的每个部分必须产生一个完整的功能测试计划。对于安全和其他任何部分的设计也是这样。如果传输层安全协议(TLS)将作为选项添加到网络协议栈中,测试计划必须包括测试(使用和不使用 TLS 的),每个案例有一个通过条件。安全的设计总是意味着安全的测试计划。

一个组织如何培训人员,使他们能够完成这些困难的架构任务?软件安全专家 Gary McGraw 说:

多年来我一直挣扎于如何教导人……安全设计。唯一真的起作用的技术是学徒制。简短的学期,就可以帮助深刻理解安全设计原则。^[14]

McGraw 的言论暗示,为了建立一个高素质的团队,每个组织只能在有足够能力和经验丰富的从业人员身上投资,这些人员可以指导和教他们做什么,或者雇用可以提供适当引导的顾问。这些都不可能是很廉价的。写作本书时,仍然缺乏熟练的安全架构师,比那些有能力和意愿传授自己所知给他人的人组成的小团队人数还少。SDL 所需的架构和设计技能需要时间来磨练,需要时间来发现关键的领导者,然后需要时间让这些领导者从手边那些可用的和感兴趣的人那里建立一个需要技巧的实践。在这样一个长期的指导下,甚至那些积极的初级人员只要两到三年就可以完全独立地工作,并开始在自己的方向上处于领导地位。这是一个效果显著的时间投资[⊖]。在上面引用的博客条目中,McGraw 引用了 Salzer 和 Schroeder 在 1975 年开创性的论文“计算机系统的信息保护”^[15]作为架构师一系列原则的起点。这些也可以用作训练和指导的基础。McGraw 的原则是:

1. 保护最薄弱的环节。
2. 纵深防御。

⊖ 请注意,并不是每一个人从开始训练都将有能力和动机来完成。我们的经验是,其中 1/3 ~ 1/2 的这些人不会成为安全架构师。

3. 安全地故障。
4. 授予最低权限。
5. 独立的特权。
6. 节约机制。
7. 不共享机制。
8. 不愿相信。
9. 假设你的秘密不安全。
10. 完全协调。
11. 安全使用。
12. 提高隐私。
13. 物尽其用。^[16]

深入讨论这些原则不是这项工作的初衷。安全从业人员可能已经熟悉若非全部也必绝大多数原则。我们引用这些原则作为一个如何成功实现一个成熟架构实践的例子。不管你选择采用什么原则，架构模式将会出现。例如，当我们考虑一个经典的问题时，你的心中要有“不愿相信”和“假设你的秘密不安全”的想法。当一个评估员第一次遇到存储在永久存储上的配置文件时，能考虑到日常读取和解析文件是一个受攻击面可能是很让人惊讶的。一个是不禁想问：“难道这些不是程序私有的吗？”不一定。评估员必须考虑什么样的保护可以用来防止攻击者利用配置文件来传输漏洞和有效载荷作为攻击途径。至少有两个安全控制：

1. 仔细设置配置文件的权限使得仅可以让目标应用程序读写配置文件。

2. 在程序中，不管任何目的，在使用输入数据之前必须严格验证所有来自配置文件的输入数据，建议对这些输入数据进行模糊测试以保证数据的有效性。

一旦遇到过一次，可能几次攻击之后，这两种模式成为一种标准，评审员将开始捕捉威胁模型每一次的攻击。这些模式开始看起来“零碎的，干巴巴的”。^①如果配置文件统一直被跨文件夹使用，那么可以从这些模式里诞生一个标准。原则导出模式，然后可以被标准化。

这些权威断言产生特定的模式，并产生一些特定类型的控制，这些控制将适用于这些模式。这些模式可以被不相关的系统应用。由于架构师汲取经验，他们将可能会写出其模式可以适用于特定类型的所有系统的标准。

为了捕捉微妙的变化，最好的方式是同行审查。如果对评估人部分有任何疑问或不确定性，就要建立一个评估同行审查的系统或威胁模型。

使用基本的安全原则作为起点，加上强大的指导，安全架构和设计技能最终可以形成。你还需要的技能是用于计算风险的方法。

一般来说，在我们的经验中，信息安全风险^②还不是很好理解的。威胁成为风险；漏洞等同于孤立的风险。通常情况下，在任何可能的一系列情况下，对任何系统最坏的影响是假定的。这样做，而不是仔细调查在何种威胁下，一种特定的漏洞是怎样暴露出来的，如果这样做，那么导致溢出的因素是什么？我们已经看到过非常耐用的服务器场设备，这些设

① 过于标准化也有危险：评估员可能从开始就忽略标准之外的微小之处。在可预见的未来，评估和威胁建模将继续成为一种艺术，这种需要人类的智慧才能彻底领悟。小心尝试规范化一切的诱惑，因此，试图把专家从过程中剔除，虽然这可能是一个诱人的愿景，但是它会导致遗漏漏洞。

② 信息安全风险计算已经超出了本章的范围。请参阅 Open Group 的 FAIR 方法。

备被仔细看管以限制许多常见的 Web 漏洞的影响，以至于部署那些漏洞的风险非常有限。风险计算的每个部分都必须考虑；但是，在实践中，我们发现，在计算风险时，整体的方法没有被采用。

一个成功的软件安全实践将花时间培训风险评估技术，然后建立或采用一种方法，这种方法是轻量级的，足以快速且频繁执行，但足够使决策者有足够的风险信息。

9.5 测试

设计和编写软件是一门创造性和创新性的艺术，并且也包含相当多的自律性。当尝试写出有安全保障的无漏洞代码时，创造性和自律性之间的矛盾是尤其真实的。错误是无可避免的。

SDL 的安全性测试方法的彻底性是关键，测试是 SDL 纵深防御的关键所在。测试方法必须重叠。由于没有一种测试方法可以满足所有要求的安全保证，所以用彼此重叠的方法来帮助确保完备性和对所有代码以及会蠕变的漏洞类型进行包装。图 9-11 描述了 SDL 中测试方法的高层次类型。根据经验，测试方法也是有缺陷的并且可用的工具远远说不上完美。不把安全保证这些“鸡蛋”放在一个篮子里是很重要的。

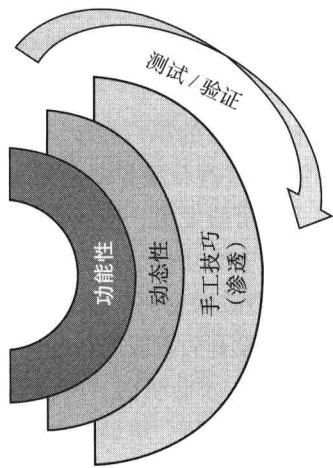


图 9-11 完整的测试套件

基于一个涉及许多不同类型且使用许多商业和免费工具的项目中的大量用例，大多综合的商业工具不是那么容易学习、编辑和使用。因为测试人员对于一部分可用工具中变得精通，可能有一个趋势，依赖每一个测试员的工具专业知识，而不是建立起一个全面的项目。我们已经看过一支小队用定制的方法（攻击和渗透），而下一支小队却使用特定于语言和平台的免费工具，旁边就又有这么一支仅做数据静态分析或使用动态工具的小队。这些方法中任意一个都不是完备的，都很有可能错过重大的问题。下面是关于对合适的问题集如何使用合适工具的一些建议。

如前面所指出的，我们相信静态分析包含在 SDL 核心中。我们更喜欢把它用作开发者编写代码过程中的一部分而不是正式测试计划中的一部分。请看 9.1.3 节来获取更多关于在 SDL 中使用静态分析的信息。

9.5.1 功能测试

安全功能和控制的每一个方面都要先测试以确保它们按照设计正常工作。虽然这是显而易见的，但是在任何一个 SDL 中加入安全性测试计划都至关重要。正如项目组和人员所说，既然安全功能没有专门包含在测试计划中，就没必要测试，所以也没必要执行。这种说法和安全专家的想法：就只执行计划中的任务，一样不负责任，而其他人员则创造性地去掩盖那些必须的需求。制定 SDL 任务表时，声明对每个人都很有用，每一步都要明确制定任务。

功能测试套件是架构和设计之后的必要工作。必须全面地证明每个安全需求都已设计然后才能搭建实现。安全或者其他方面的功能都必须明确地测试，以保证工作和设计相吻合。描述功能测试集不是本章讨论的问题。但是，我们建议使用多种方法构建可靠的校验体系。

- 和预期一样吗？
- 测试的极端和边缘情况以证明这些得到妥善处理，不扰乱预期的功能。
- 试验滥用的例子；对于输入，这些都将模糊测试

基本上，这些测试必须检查所指定的确切行为。就是说，把规范变成一个测试用例：当用户尝试加载受保护的页面时，有没有验证要求？当输入正确的登录信息时，要求满足吗？当输入无效凭据时，拒绝验证吗？

边缘情况是最困难的。这可能是默认行为的测试或者未明确指定的行为的测试。在验证的情况下，如果一个页面没有选择保护，Web 服务器是默认的吗？如果保持配置默认，尝试两种二元默认：无页面保护对比所有页面的保护。

这个例子的另一边缘情况，可能用无效的用户和垃圾 ID 来测试验证，或试图重放会话令牌。会话令牌通常有一个超时。如果浏览器和服务器的时钟不同步会发生什么？是令牌仍然有效，还是已过期？这对普通用户都可能发生，但通常不会。

最后，许多功能将被攻击，特别是安全功能。换言之，任何可滥用的地方都可能被滥用。攻击者会登上一个登录页面，试图暴力破解密码。测试计划不仅要包括任何锁定功能测试，还应该能发现任何处理多个快速登录而没有失败或崩溃的应用程序或认证服务中的缺陷。

根据我们的经验，最有经验的高端人才会明白“符合设计”的测试，以及边缘情况。但是滥用案件可能是一个新概念，它将需要支持、培训，也许还有导师。

9.5.2 动态测试

动态测试是指执行源代码并观察它如何执行特定的输入。因为所有活动都在这类程序的执行中验证，所以动态测试是至关重要的。^[17]

动态测试是执行程序的测试。在安全上，动态检测一般从攻击者的角度来部署。从长期纯粹的意义上说，在执行的程序上的任何测试是“动态的”。这包括漏洞扫描，自定义代码漏洞扫描（通常称为应用程序扫描），模糊测试，以及任何形式的攻击和渗透测试。下一节讨论 SDL 中攻击和渗透测试的地方。由于所需的技能和通常的费用，我们已预留攻击和渗透测试作为一个特例。不过，我们鼓励任何能进行大规模攻击和渗透的组织做尽可能多的测试。在现实世界的测试中，熟练的攻击和渗透测试人员总是胜过 Web 漏洞扫描器的结果与典型的模糊输入。对于大多数组织而言，雇用熟练从业者的费用和它们不能在多个组织和多个项目中规范，在某种程度上阻止了攻击和渗透测试。

动态分析基于系统的执行（二进制码），通常需要用到工具。^①动态分析的优点如下。

- 具有检测出静态分析检测不到的依赖关系的能力，例如，动态依赖使用反射依赖注入等。
- 允许收集暂时性信息。
- 可以处理正在运行的数值。
- 在运行时环境上分析漏洞。
- 可以使用自动化工具来提供扫描的灵活性。
- 允许分析你看不到源码的应用程序。
- 可以识别在静态分析中漏报的漏洞。
- 允许验证静态代码分析结果。
- 在任何应用程序上都能执行^[18]。

1. 网络扫描

漏洞扫描器通常分为两大类：一种是签名测试针对支持应用程序执行的运行时间，另一种是专注于自定义应用程序代码的扫描仪。前者通常应用于基础设施，以确保适当的修补和配置定期完成，并保持最新。这些应用程序将被托管在内部（如本章前面所述），运行时漏洞扫描器对于保持基础设施安全状况维持至关重要。这种扫描仪也可以运行在一个设备类型的项目上，其默认配置没有漏洞^②。这种类型的漏洞扫描器也可能被用来对付正在运行的设备，保证设备提供了一个适当的计划部署的安全状况。然而，对于许多不属于这些类别的软件项目，一个运行时漏洞扫描器在测试计划内的价值可能有限。

如前所述，对于 Web 服务器，自定义代码或应用程序漏洞扫描器是必备的。如果软件中已经包含了 Web 服务器，应用程序漏洞扫描器将帮助查出这类问题，即攻击者可能会对自己感兴趣的 Web 应用程序的代码进行攻击。我们相信，每一个内容块，无论是动态还是静态的，都通过软件的 Web 服务器提供服务，这些 Web 服务器必须要经过应用程序漏洞扫描器的测试。考虑到今天网络攻击的水平，如果 Web 服务器暴露在公共网络上，应用程序不经过扫描是无法运行的。攻击是肯定的；只是时间问题。目前的估计是，一次攻击最短将只有 30 秒，而最长不会超过 8 小时。攻击者非常迅速地利用新发现的 XSS 错误和类似的错误。这正是 Web 漏洞扫描器关注的各种错误。

然而，Web 漏洞扫描器的有效使用，并不是一个微不足道的操作。技能与特定工具是必备的。同时，十分熟悉如何结构化和构建 Web 应用程序可以帮助测试人员优化工具中的测试套件。进一步，熟悉典型的 Web 漏洞将提供两个方面的帮助：第一，适当的测试套件配置应该在保障应用程序和它的预期用途的基础上；第二，扫描的结果可能需要合乎要求。有两个值得一提的问题。根据应用程序、代码和工具，结果可能是“噪音”。也就是说，可能存在必须被删除的误判。我们已经看到很少使用不考虑误判损失的和误判率高的 Web 漏洞扫描器。此外，在撰写本书时大多数 Web 漏洞扫描器尝试每种类型问题的变体。大多数工具将作为另一个漏洞报告每一个变体。尽管报告多个漏洞，但所有的变体可能源于代码中的一个 bug。

① 这里，工具指为了分析而向代码中添加的诊断消息和停止点。

② 这是因为一个应用程序可能会包括一个运行时栈，包括支持操作系统及其服务。除非整个操作系统是专有的，否则可用的工具可能包括适当的漏洞签名来进行测试。在一个完全专有的操作系统的情况下，可能需要建立专门的工具。

对于许多工具，在漏洞和实际编程错误之间存在多对一的关系；一个输入验证错误可能会产生许多“漏洞”。因此，测试人员和 / 或编程团队需要限定结果以找到实际的编程错误。

对于任何将运行网络漏洞扫描工具的测试人员需要培训和实践。像静态分析工具一样，当网络漏洞扫描工具简单地加入项目团队时，其结果可能会令人失望。相反，我们不止一次发现，一个成功的方法都是从小处和有限的地方开始的。选择可用于试验的项目，这将从网络扫描那里获得益处。发现对安全测试感兴趣的人，甚至希望提高他们的职业可能性。然后，工具的漏洞测试套件工具减弱到只有制造商才认为测试提供极高信心——实现超过 80% 的结果，也就是说，只有不到 20% 的误报率。我们甚至开始只使用误报率低于 10% 的那些测试套件。

这样，测试人员将积极学习工具，工具会产生高信任度的结果，可以依靠这些工具找到真正需要修复的缺陷。在这个过程中每个人的信心和工具最终将会显著提高。从这个强大的地方开始，测试人员和开发团队会更加愿意尝试他们可以容忍多大的误报率，这仍然得到有用的结果。不同的项目需要找到自己的平衡点。

从这些有限、试行的起点，推出网络漏洞扫描的团队将获得关于起作用的、不起作用的、可能会遇到阻力的重要信息。再一次，我们反对只使用委托工具，然后没有任何经验和实验，没有适当的训练补给，就置之不理。我们看到了太多的程序在这种方式中挣扎。相反，从小事做起，扩大之前取得成功和自信。在做任何特定的测试需求之前，一个好的临界点将收获 60% 的自愿参与度。

2. 模糊测试

模糊测试或模糊是一种黑盒软件测试技术，它的主要功能是用自动化的办法注入畸形数据来寻找实现上的 bug。^[19]

由于非 Web 输入的多样性，找到一种能够适用于各种输入方式的工具是不切实际的。有的开发团队可能会编写他们自己的测试工具。然而，为了将这种策略作为一种安全战略，工具设计者和使用者必须对已经知道的各种输入攻击有大量的知识。而由于定期更新以应对新的攻击方式、实现策略转移、改变现有的攻击方式，因此所有攻击场景必须跟着进行更新。这正是工具厂商做的。这样的策略对大多数组织可能是不切实际的。

许多攻击方法通过模糊测试被发现，即使用针对输入攻击层面的模糊工具。一旦一次意外的反应是从程序的输入获取的，攻击者（或研究员）就可以检查有问题的输入和程序的行为来判断该漏洞是什么，以及如何最好地利用它。

值得庆幸的是，软件测试人员不需要很深入地探索这个。如果输入产生不正确的行为，那么该程序没有充分防御：发现了一个 bug。需要理解这种情况。这正是模糊测试的重点：一个输入导致一个不正确的行为。对于软件，特别是安全软件，必须能够优雅地处理输入的任何数据序列。如果没有很好地处理不适当或不期望的输入将会让用户关注度最小化。更进一步说明，这个程序存在漏洞。

模糊测试程序中的每个输入就是描述需要被测试数据的范围的问题。大多数模糊测试工具处理多种不同类型的输入。测试仪设定了输入的类型和系列。在模糊工具限定范围内随机化，连续发送不正确的输入，就像可能是在搜寻漏洞的攻击者。

模糊工具会自动尝试许多变化的输入过程。配置文件可以模糊处理，命令行界面可以模

糊处理，API 可以模糊处理，Web 服务、网络协议等都可以模糊处理。其实，任何类型的输入，包括 Web 服务器，都可以模糊处理。由于有大量可用于在应用程序中扫描 Web 服务器的工具，因此我们把重点放在对其他类型的输入进行模糊处理。如果一个机构能够研发出功能强大的模糊工具，没有理由不让这个工具对每个输入（包括 Web 服务器）进行适配。根据我们的经验，它能够区分可以扫描的 Web 输入和其他没有直接扫描工具的输入。因为其他输入没有更加模糊的工具来适应它。

模糊测试是一个间接类型的输入验证，而漏洞扫描器高度集中于已知的攻击方法。一个完整的安全检测程序会识别每种技术的适用性并适当地应用它们。有一些重叠；某些 bug 会同时响应这两种类型的工具。

9.5.3 攻击和渗透测试

攻击和渗透测试 (A&P) 一般是由像最熟练的攻击者一样的测试人员完成的。这些测试人员会侦查目标系统，识别攻击表面。同时这些测试人员将利用那些专家级的攻击者常用的工具去识别那些明显的错误以及隐藏在系统中的微妙的逻辑错误。相比较而言，逻辑错误是最难以辨别的。所有错误（除了最简单的逻辑错误）都需要人去识别。

由于攻击和渗透测试耗时耗力，所以我们将其分离了出来。渗透测试人员获到高额的薪酬总是有原因的。要得到高质量的结果是需要极高的理解力和技巧的。Alan Paller 曾不经意间说当今熟练的渗透测试员不过 1500 人左右。我们不知道到底多少，但是目前解决这类工作所需要的高技术测试人员是极其匮乏的。当前的情况可能在未来一段时间都不会改变。由于这类人员的稀缺，所以我们建议只针对那些可能面临严重攻击的核心部分和主要发行版进行攻击与渗透测试。

如果你的组织拥有充足的攻击和渗透测试资源，而熟练的人力因素是安全方面最强大的测试力量。任何可以测试的都应该被测试。然而，在我们的众多调查报告中我们可以看到测试人员大多没有这方面的能力，没有花足够的时间去理解测试目标、运行默认测试和报告数以百计的漏洞。这类测试远没有达到预期目的。开发团队可能看到开始的几个漏洞，声称它们是误报的，然后停止测试。这是对充满可能的漏洞（而不是真实问题）的报告的典型回应。一般来说，在这些测试中，攻击测试没有调整到正确的目标或者可能这个目标和部署时的配置不一致。而这时，所做的一切都只是浪费大家的时间。

相反，应该将你的时间和精力放在最具价值的目标上。不能出现问题的关键代码从攻击和渗透测试那里受益很大。在主要版本发布前或者重大修改后我们都能从中受益不少。这就是我们建议的可以从熟练的攻击与渗透测试中获取最大受益的地方。

由于一次攻击和渗透测试会消耗很长时间，所以当进行高密度的测试时一定要充分考虑代码的修改频率。当这个修改（更新）频率高于测试频率时，可能在测试完成前又引入了新的漏洞。为了得到最佳的结果，这两个因素一定得做好权衡。一般来说，即使每天都有更新，但是不会有大的发行版，更不会有主要修订版的出现。因此，对核心关联度较大的代码段进行测试是一个很好的投资。

那什么是关键代码呢？我们已经看到过很多关于“关键”的定义：

- 最高收益系统
- 最易受攻击系统

- 最大的系统
- 投资最大的系统
- 最具战略的系统
- 监管最严苛的系统
- 风险最高的系统
- 处理最敏感数据的系统

以上每一种定义都可以由其他的系统定义轻易地推翻。一个可行的办法是让商业领袖或者组织者决定那类系统是关键。多种因素可能都得考虑在内。以上没有一种定义是相互排斥的，不同的因素可以给系统的关键性增加权重。我们建议一种开放的方式。如果组织可以承受一个更大的网，那么从长远看它会更好。一个组织不想因为仅仅一个关键性因素而错过一个重要的系统。

9.5.4 独立测试

可能有的情况下，利用第三方安全测试是有益的。如果客户对产品的安全性特别敏感，在购买之前，他们可能需要一个有关系统安全健康的第三方认证。

在顾客对安全性证明的需求显而易见的情况下，我们已经使用的一个成功方法，就是把第三方认证作为已售产品的直接成本。当系统在没有第三方认证的情况下不能出售给很多客户时，第三方认证被认为是为这些客户建立系统的成本的一部分。一份认证报告通常可用于许多客户。

事实上，有时获得关于系统的独立视图也有好处。正如所有人类的努力一样，如果评估方过于熟悉系统，他们可能会错过重要的因素。运用一些独立的分析将会让关注点集中到真正的问题上。

“独立”并不一定意味着完全置身组织外部。我们只引入一名没见过这个系统而且不知道这件事的安全架构师就成功了。如果组织够大，通常有资源负责可用来检查工作的替代系统。

值得再提的是，安全架构师拥有的最强的工具之一是同行审查。工作中很容易错过一些重要的东西。我们在工作中已经制定了多个组织中同行审查的机制，例如，分析中出现的的不确定的问题需要一些有经验者达成共识来解决。这样，每个评估员可以让他或她的工作得到检查和验证。

如果这里列出的任何测试方法中有任何的不确定性，获得一个独立的视图可以帮助验证该工作或发现该方法中的任何漏洞。

9.6 敏捷：冲刺

我们认为，使用敏捷过程生产安全软件的关键是将安全集成到来自于架构的敏捷过程中，该架构贯穿着测试。而不是将瀑布开发强制置于敏捷过程之上，安全必须是敏捷的；安全从业人员必须放手刚性流程，并进入是敏捷开发精髓的对话和协作中。认识到，我们必须相信敏捷开发团队并和该团队一起工作，同时利用敏捷过程，而不是抵制敏捷过程及其从业人员。

图 9-12 演示了，在这种 Scrum 情况下，原型 SDL 如何显示图 9-1 中的变化，以反映敏捷过程。需求和架构是敏捷周期或“冲刺”的前端流程。架构加入反复的冲刺周期中。在一系列的冲刺结束时，运用抢先测试。SDL 中的所有其他任务发生在每一次冲刺中。

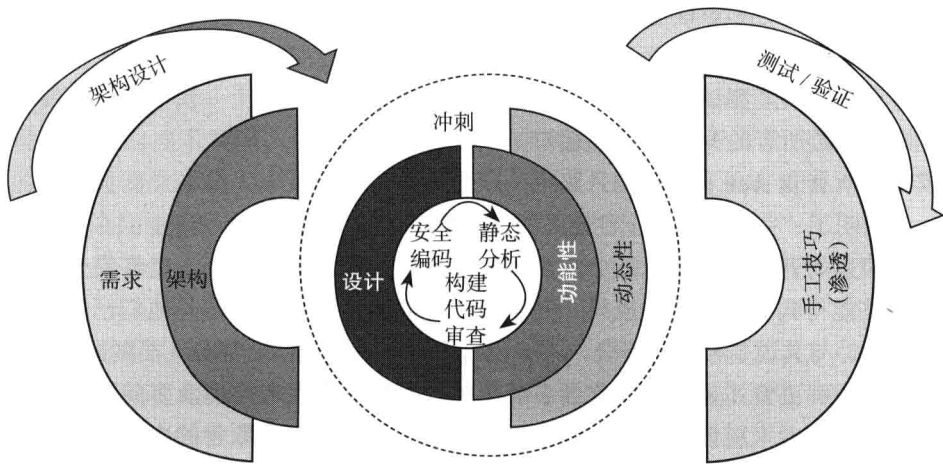


图 9-12 敏捷 SDL

一轮冲刺是一个开发周期，在该周期中建立代码块（用户故事）。每个 Scrum 团队选择团队冲刺的周期。典型地，冲刺持续时间大约为 2 ~ 6 周。每个冲刺周期正好一样长；这允许实现开发的节奏。在冲刺中任何未完成的项都会被放入待建的未完成项清单。在冲刺开始时，一些设计工作将发生；至少充分的设计需要进行，以便开始编码。尽管如此，设计可以改变一个冲刺展开。此外，在冲刺期间一有要测试的东西，测试就开始了。以这种方式，设计、编码和测试并行进行。在冲刺结束后，团队将在持续的改进中检查该周期的结果。

在 Scrum 中，在用户故事创建的过程中考虑要建立什么。这就是“早”的一部分。与产品所有者[Ⓐ]的密切关系对于将安全用户故事写入未完成清单至关重要。这种关系在未完成清单优先级化的过程中也十分重要。在冲刺计划会议期间，将项目引入会议来构建，大量的设计被确定下来。要么通过直接参与要么通过委派代表，使安全性成为过程的一个组成部分。

安全专家需要让自己参与进来，在整个冲刺阶段来回答有关实施细则和安全地建立用户故事的正确方法的问题。让我们的设计出现。让设计安全地出现。作为 Scrum 团队中受人尊敬的一员，早期发现安全缺失会被赞赏，而不是抵制。设计和实施被改进：设计大多数情况下将更加正确。

优先级对话的一部分必须是“给定通常的时间和预算限制，可能建立什么”和“必须建立什么以满足目标、安全性和其他要求”之间的相互作用。安全专家参与进来，不是以“唯一正确的方法”，而是以一种态度：“我们如何共同完成这一切，并做得足够好，以满足要求？”

最后，由于编写安全代码更需要自律和实践，因此适当的测试和漏洞保证措施需要作为每一个冲刺的一部分。我们认为这些需要作为“完成”的定义的一部分。根据前面讨论的七个确定问题，所提到的“完成”的定义可能类似于以下概念：

完成的标准

- 1. 手动检查过所有的代码（并且修复所有漏洞）。
 - a. 同行审查过所有代码

Ⓐ 产品负责人在 Scrum 是一个正式的角色。产品负责人需要采纳客户和用户的观点。她或他负责创建用户故事和优先级。产品所有者可能是一个独立的开发团队的成员，一位资深架构师（尽管一般不是）或产品经理，或者类似的人员。他不应该是职权在开发团队成员之上的人。

- b. 资深或者安全专家审核过核心代码。
2. 静态分析过所有代码。
3. 通过了所有功能测试。
4. 动态测试过所有的 Web 服务器接口（并且修复所有漏洞）。
5. 模糊了所有非 Web 程序的输入路径（并且修复所有漏洞）。

每一个指明了“完成”的安全性定义的条目都将在相关部分具体描述。

在一个快速的开发过程中，每个人都和安全息息相关。禁止安全人员将安全问题抛之脑后还期待最终能得到一个安全的成果。开发团队可能会认为一个问题是他们已完成工作中的一个插入部分。与其说是合作的安全，不如说这个结果是敏捷团队部分人员的阻力。

更加有效的解决方式是将问题拿过来解决掉。要把安全性考虑到众多问题和重要项目的总体目标中。安全专家应该牢记，在开发过程中总有需要做出有取舍的决策。要想取得所有项目（包括安全方面）的成功，合作团队不仅仅需要把安全性放在其恰当的位置进行考虑，还要更多地发掘创造性的解决方案。安全人员不惜弄脏自己的双手，将虚拟的指甲里面粘上油脂就是为了获取 Scrum 团队的信任和支持。

当然，所有开发团队都会面临的一个问题就是对即将进行的工作设置一些优先级，对于安全类项目尤其如此。一个鲁棒的、基于风险的方法将有助于把实际的影响纳入优先级方程式中。

基于我们的经验，如果敏捷团队拥有足够的风险指导来了解到：针对其他任务来说，安全项拥有更高的优先级，那这个团队就可以做出更好的决策。有一个经典的例子：“这是一个可能导致损失的漏洞，得立马修复它”，但这长期以来都没能影响到决策者。应该避免忧虑、不确定性以及怀疑。

相反，不需要考虑信息安全方面损失的可能性，而应考虑实际业务对系统的影响，因为它将使用预期的部署来运行。关注影响的方式将进一步有利于合理决策的制定。安全队列中可能有一些条目因为其他重要特性而推迟。团队的信任建立在包括安全专家在内的每一个人员的充分参与下。当一个安全专家成为敏捷团队的一部分时，每个人都取得了胜利，安全也得到了保障。实际上，当安全问题得到慎重考虑并且所做决定是基于风险情境时，安全具有如下两大胜利的标志。

1. 安全成为决策结构的一部分并且占据一席之地。它不再是一个附加的东西。对于安全的考虑成为构建软件的一种心态而不是外在附加的东西。

2. 如果风险得到了慎重的考虑，那么高风险的条目应该优先考虑。摒弃每个安全问题都是同等重要的想法可以让那些真正危险的条目（极为罕见）得到应有的重视。那些风险较小的条目可以根据相对重要性进行适当的安排。

对于大多数机构来说，他们拥有的娴熟安全专家数量少之又少。因为数量太少，所以不是每一个 Scrum 团队都可以拥有自己的安全专家。通常一个安全专家负责好几个团队。这些专家需要将自己的技能按照时间分片分给不同项目。需要给这些人员分配不同工作的切换时间来保证他可以在各个工作中切换自如。优秀的项目管理技能是必需的。我们可以通过安全专家或者项目管理实践实现这一点。项目经理有助于调度和交付那些使矩阵分配策略更加有效的任务。

然而，多个项目只分配一个安全专家是极其危险的。最重要的是，工作量过重实在是太

常见了。如果给一个安全专家分配太多的项目，他就不得不去研究自己该如何拥有足够的时间去处理下一个项目，同时还要在各个项目切换之前重新梳理一下每个项目的具体内容。这样的话，其实没有任何一个项目真正完成了。我们不仅没有时间去培养更多的安全专家，而且如果没有持续的技术发展，我们就不会有安全专家的发展。如果安全专家已经不堪重负，那么他的研究时间就将会大大地减少，从而导致整个项目的崩溃。更有甚者，如果安全专家没有时间来认真彻底地研究问题，他将做出很多不好的决策。

避免超负荷工作；我们需要善于观察细节，因为每个人对于任务量过多的定义都有着不同的阈值。

太多的超负荷工作将导致项目被迫推迟，因为整个团队都在等待安全专家的参与。安全专家将成为团队速度的瓶颈。或者，更加糟糕的是，设计和实现就不会在所需安全专家和帮助下开展。这样总会导致产生出不安全、有安全逻辑错位或者高价返工的产品出来。这些正是敏捷过程中经常会遇到并需要解决的问题。

实际上，一些超负荷工作的安全人员可能会以一种不恰当的参与方式来弥补那些被抛出的象牙塔宣言。敏捷过程的标志和本质是协作与信任，然而，这两者却被安全人员认为非高效而被忽略了。如果没有足够的时间建设团队合作意识，安全任务过重的人将没有足够的时间去很好地融入每一个 Scrum 团队。如上所述，当安全人员明白所有的问题必须由整个团队共同解决时，安全就拥有了一席之地，而并不仅是一个附加部分。正是那些拥有安全专家的敏捷团队中的安全项目，像敏捷团队功能和自治中的附加部分一样，正在试图停止。

然而，用足够多的安全人员来调度虚拟团队完成多个项目还是有可能的。我们已经多次见证过这些项目成功完成，前提是避免超负荷工作。事实上，我们应该记住，团队精神需要时间和经验来培养。在形成和学习的过程中总会产生错误与失误，但这些是取得成功的必经之路。

9.7 成功的关键因素和度量标准

这一章不是这本书早先讲到的 SDL 框架阶段的一部分，相反，它只是第 3 ~ 8 章应用到现实世界情境中的概要信息。因此，本章并没有给出特定的成功因素和标准。下面描述的成功因素是从 SDL 在实践中的应用这个角度来讲的。

9.7.1 安全编码培训计划

对于组织运行模式来说，建设一个安全编码计划是十分必要的。它应该为多样化的利益相关者服务，而不只是仅仅为了服务研发工程师这些人。一个有效的安全编码计划会帮助产品经理了解到这些实践，有助于给实践中的架构润色，以及给工程师提供如何编写良好代码的指导原则。计划的模板也应该适用于一些特殊语言（例如，Java、C、C++）。Web 服务体系结构和密码的适当使用也必须是所有有效安全编码培训计划的一部分。

9.7.2 安全编码框架 (API)

除了安全编码之外，开发者应该注意能用来作为编码一部分的任何安全的 API 和存在的可用安全编码框架。这可以预防临时编码实践来解决众所周知的安全问题（例如：预防 XSS）和规范代码——随着时间的推移标准化有助于保持代码的可维护性。不同类型的安全测试将

会指出由于不恰当的使用或者没有使用安全代码框架（API）所导致的缺陷。应该定期开展安全编码的训练项目，用以增强实践，并不断更新它来弥补由于 API 的不正确使用所导致的缺陷问题。

9.7.3 人工代码审查

每一行提交构建的代码都需要进行同行审查，无论是多么小的模块或者是一小组变化，也就是所谓的最佳实践的“卫生习惯”。同行的审查不应看作是一种批评，而是一种使代码更加可靠、易读、安全的方式。关键代码应该由哪些精通安全编码实践和算法实现的人来审查。根据我们的经验，多次代码审查应该是非常有益的。

9.7.4 独立代码审查和测试（专家或第三方）

一旦编码完成，第三方或者独立的专家应该给出一次全面的编码审查。这里的目的是捕获在 SDL 构建和测试活动中没有发现的问题。独立审查和渗透测试中的发现也应该被参与静态分析和编码审查的人们所知晓。在下一轮开发中，很多这些问题都会在同行审查或者测试计划期间通过静态分析的方式来进行处理。

9.7.5 静态分析

在代码检入进行人工审核之前，应该通过静态分析找到常见的错误和违反最佳安全原则的实践。这将能减少在人工审查和专家审查阶段添加的评论数量。在检入代码之前进行静态分析的行为，应该是强制性的、理想的，应该紧密集成到开发环境和开发者工作流中。

9.7.6 风险评估法

正如第3章和第4章所讲的一样，对于一个成功的 SDL 项目来说，有一个风险评估框架是必需的。威胁建模和架构评估进入到 RAF 中。RAF 给风险状况设置优先级并基于风险的严重性和影响做决策。

9.7.7 SDL 和 SDLC 的集成

关键性的问题必须进入到允许 SDL 的 SDLC 周期，也就是说，软件安全问题，很容易集成到项目和开发实践中去。作者再三强调创建一个 SDL 项目并把它映射到 SDLC 周期，而不是使安全需要成为一个项目计划行条目。没有合适的指导方针，按项目计划的团队很有可能编造他们自己关于安全活动很少的解释。

9.7.8 架构人才的发展

安全的构架人才并不容易找到，因为它位于技能金字塔的顶端。在担任架构师的角色之前需要有软件开发以及在数个不同的安全领域的背景知识。这就意味着常常缺乏有准备充分的、有能力的架构师候选人。项目所在地的发展和导师资源将是关键性的，最终它们将承担安全架构师的角色。从外部招聘架构师也往往是不太有用的，他们需要花费大量时间来理解这个组织的软件、环境和每个组织特有的实践行为，然后才能够应用现实指导原则。一个架构师指导的项目将会在时间方面有明显的投资回报。

9.8 度量标准

正如早先提到的，这段并不属于 SDL 阶段的一部分，但以下列出的度量标准同样适用于第 3 ~ 8 章。这个列表是实用度量标准的汇总，来源于作者现实生活中对于 SDL 应用的经验之谈。

- 成熟的安全编码方案
- 在编码中使用审查 API 的百分比
- 软件代码检入时人工审查的百分比
- 软件代码检入时人工审查的代码行数
- 在专家审查阶段，人工审查出现失误的百分比
- 在人工审查阶段，静态分析出现失误的百分比
- SDL 阶段需要着重调整的团队审计软件的数量
- 运用自己集成开发环境（IDE）中集成的静态扫描软件的开发人员的百分比
- 人工审查、静态分析和专家审查阶段发现问题的数量
- 内置于 SDLC 的 SDL 阶段百分比
- 组织中“未来”架构师的数量
- 被评定为生产系统的软件系统所占百分比（你完成了每个项目和每个系统吗？）
- 设计生产的安全审查完成百分比
- 成熟的安全设计审查过程
- 基于设计审查建议的异常数量[⊖]
- Web 网络动态分析所覆盖 Web 服务器所占百分比
- 输入模糊化所发现的问题数量
- 接收 A&P 独立测试的主要发行版本的数量
- 外部报告的安全漏洞数量

9.9 本章小结

综上所述，正如我们所看到的，安全任务规划和安全软件构建构成了应用的安全开发生命周期。为了使这些任务更好地执行，开发团队成员之间的关系，安全人员和开发团队之间的关系决定着成败。

在整个 SDL 交付完成软件期间，需求收集和后续思考很早就开始进行了。架构和设计时参与旨在构建正确和适当的软件安全机制，这就要求测试计划的编码正确，并且功能正确，无漏洞。测试计划包括多种方法，具体采用哪种方法需要依据软件暴露的攻击表面而定。安全是一个端到端的过程，通过任务之间的相互重叠来提供连续性和保证。至今还没有一个好的方法可以让某个任务解决整个安全问题。

当安全人转向开发者的关注点时，安全呈现出的不是漏洞，而是需要融入成功的属性，以及开发人员希望删除的错误，开发人员和安全人员可以团结一致工作以开发出适当的安全软件。安全不是作为非功能性的、自上而下的命令。相反，完整软件中那些值得优先考虑的

[⊖] 不要过于依赖这个看似无辜的度量。熟练的安全架构师经常使用异常从“是否修复”转向“什么时候修复以及如何修复。”

属性中，安全需要得到合理的重视。在整个 SDL 中，适当的优先级是安全主题专家通过深度和积极参与获得的。

事实上，并不是每个安全活动都适合所有项目。项目因为大小、临界值和范围的不同而有所区别。一个可行的 SDL 会以这些差异进行划分。我们提供了 7 个确定的问题作为将正确的安全任务分配给合适项目的标准。

姑且不去讨论所列出的具体问题，建立一套可以为每个项目确定适当安全活动并且特定于项目的问题，不仅能确保每一个项目安全正确地执行，而且可以回答项目管理中的典型性问题：

- 我们需要做什么？
- 我们必须执行的最少活动是什么？

以一个简单和容易理解的方式提出你的 SDL 确定性问题，将在安全团队和项目团队之间建立信任。最终，信任是构建安全软件的关键，就像尽可能多地培训安全编码和严格的安全测试一样。为使安全能够面向设计，必须从整个生命周期开始时就考虑安全性。必须编写代码来最小化漏洞并且保护攻击表面。程序的功能和不存在漏洞必须由一系列测试来验证：架设→设计→编码→测试。

安全软件必须满足以下几方面。

- 没有可以被恶意操控的错误，即没有安全漏洞。
- 包含客户要求的安全特征。
- 自我保护——软件必须抵制针对该软件发布的攻击类型。
- 也就是说，不能以“失败”这样一种方式作为将成功攻击的后果降到最低的方法。
- 用明智的、封闭的方法安装。

未能满足这些需求，软件就不能成功实现其功能，就不能在这个不断有数字攻击的世界安全地存在。需要指出两个重要且环环相扣的方法：构建正确的特性让软件可以保护自己和软件的使用者，移除开发中不可避免的错误；不正确的实现（逻辑错误）必须被捕获并移除。引入的任何漏洞，必须被发现并根除。

安全开发生命周期是一个基于关系和现实世界的过程，开发团队通过该过程开发安全软件。

参考文献

1. Perlis, A. (1982). *Epigrams on Programming*. ACM SIGPLAN Notices 17 (9), September, p. 7. Retrieved from <http://www.cs.yale.edu/quotes.html>.
2. NATO Science Committee (1969). *Software Engineering Techniques*. Report on a conference sponsored by the NATO Science Committee, p. 16, quote from Edsger Dijkstra, Rome, Italy, 27th to 31st October 1969. Retrieved from <http://homepages.cs.ncl.ac.uk/brian.randell/NATO/nato1969.PDF>.
3. U.S. Department of Homeland Security (2013). Software & Supply Chain Assurance: Community Resources and Information Clearinghouse (CRIC), “Mitigating the Most Egregious Exploitable Software Weaknesses: Top 25 CWE Programming Errors.” Retrieved from <https://buildsecurityin.us-cert.gov/swa/cwe>.
4. Mitre Corporation (2013). “Common Weakness Enumeration (CWE).” CWE-434: *Unrestricted Upload of File with Dangerous Type*. Phase: Architecture and

- Design*. Retrieved from <http://cwe.mitre.org/data/definitions/434.html>
5. Open Web Application Security Project (OWASP) (2013). *Some Proven Application Security Principles*. Retrieved from <https://www.owasp.org/index.php/Category:Principle>.
 6. Open Web Application Security Project (OWASP) (2013). *OWASP Enterprise Security API*. Retrieved from https://www.owasp.org/index.php/Category:OWASP_Enterprise_Security_API.
 7. Ibid.
 8. Beck, K. (1999). *Extreme Programming Explained: Embrace Change*, p. 18. Addison-Wesley Professional, Reading, MA.
 9. Sheridan, F. (2012, June 13). *Deploying Static Analysis I—Dr. Dobbs*. Retrieved from <http://article.yeeyan.org/bilingual/334758>.
 10. Ibid.
 11. Golden, B. (2013). *What Is Systems Architecture?* Retrieved from http://www.lix.polytechnique.fr/~golden/systems_architecture.html#principles.
 12. The Open Group (2005, November). *Guide to Security Architecture in TOGAF ADM*, p. 5. Retrieved from <http://pubs.opengroup.org/onlinepubs/7699949499/toc.pdf>.
 13. Wahe, S. (2011). The Open Group, *Open Enterprise Security Architecture (O-ESA): A Framework and Template for Policy-Driven Security*, p. 5. Van Haren, Zaltbommel, The Netherlands.
 14. McGraw, G. (2013, January 18). Cigital Justice League Blog: *Securing Software Design Is Hard*. Retrieved from <http://www.cigital.com/justice-league-blog/2013/01/18/securing-software-design-is-hard>.
 15. Saltzer, J. H., and Schroeder, M. D. (1975) *The Protection of Information in Computer Systems*. Retrieved from http://www.acsac.org/secshelf/papers/protection_information.pdf.
 16. McGraw, G. (2013, January 18). Cigital Justice League Blog: *Securing Software Design Is Hard*. Retrieved from <http://www.cigital.com/justice-league-blog/2013/01/18/securing-software-design-is-hard>.
 17. Singh, Y. (2011, November 14). *Software Testing*, p. 87. Cambridge University Press, Cambridge, UK.
 18. Wolff, B., and Zaidi, F. (2011, December 20). *Testing Software and Systems: 23rd IFIP WG 6.1 International Conference, ICTSS 2011, Paris, France, November 7-10, 2011, Proceedings*, p. 87. Springer-Verlag, Berlin.
 19. Open Web Application Security Project (OWASP) (2013). *Fuzzing*. Retrieved from https://www.owasp.org/index.php/Main_Page.

集成：应用 SDL 防止现实的威胁

网络威胁来源于软件缺陷，即能够被网络攻击或者软件应用程序和系统利用的疏漏。在本书中，我们已经涵盖了一些策略，通过从源头上提供安全策略，以 SDL 最佳实践的形式帮助软件开发组织避免和减少软件缺陷^①，来作为一个有效的核心软件安全的基本要素以实施软件安全性的具体方面。

在本书中，尽管实现一款没有漏洞的产品是极其困难的，甚至是不可能的，但这应该是你的终极目标。通过应用本书中的最佳实践，你开发的软件将会尽可能远离安全漏洞。漏洞的数量越少，攻击者利用给定的应用程序的难度就会越来越大。这绝不意味着我们可以通过使用软件安全最佳实践来阻止所有威胁，但是最大化地减少攻击面是我们的终极目标，这将让我们作为软件安全专家的工作容易些，并且使我们的对手攻击起来更困难。在本书中概括了最佳实践的实现，你将能在很大程度上减轻来自非政府行为体的大多数威胁。

在本章，我们将威胁分为三大类，具体而言，分别是：战略、战术和特定于用户的。然后，我们将提供每个类别的攻击实例，并且在本书中介绍的 SDL 最佳实践的应用将帮助您开发的软件抵抗这些威胁和攻击方法。

10.1 战略、战术和特定于用户的软件攻击

既然我们已经描述了安全软件的开发实践，重要的是在这本书末尾，提醒读者使用这些实践的重要性，以防止今天的网络攻击。在引用几个行业领导者的话之后，对于以基线保护为核心的安全软件开发实践，我们将给出这种网络威胁的一个深度概述。

结构是安全软件开发的实现策略，将安全软件的开发实践融合到机构的文化中，而不是仅仅使用软件扫描工具，是十分重要的。一些公司已经发现，使用智能的安全软件开发流程，可以减少与核心任务软件相关的 70% 的漏洞。

——Jeff Snyder，雷神公司网络开发副总裁，2012

网络攻击利用系统的漏洞，诸如不正确验证用户输入，系统组件之间不一致的设计假设和非正常的用户和运算符操作等。在组件的计划开发、测试和维护阶段，由于断开和通信错误，都可能产生软件漏洞。尽管应用开发团队对于相关的业务功能十分熟悉，但是他们通常只有有限的安全实践经验。对于越来越多需要满足安全性要求的系统组件，断开和通信错误发生的可能性也随之增加。整个软件开发流程中，系统开发和最终实际用途之间，多个开发团队之间的必要交互在项目管理当中必须有所体现。项目经理必须考虑到附加通信需求、生命周期活动之间的联系和可能的应用环境与安全需要的联系。

——Robert J. Eusion，“安全与项目管理”，2006

通过推进全行业的最佳软件安全实践，我们有机会迎来提高科技生态环境系统整体安全性的机会。

——Howard Schmidt，前美国网络安全沙皇，2013

一个组织了解其应用的安全等级和对整个 IT 行业的重要性是十分关键的。研究表明，应用层担负着 90% 的安全漏洞，但是仍然有 80% 的安全开支是用于周边安全的网络层的。对于这个发现的研究表明，对应用安全项目的投入来减少整个组织对于网络安全的威胁是很有必要的。

——国家安全应用波耐蒙研究所，LLC 和安全创新的研究课题，2013

战略攻击主要针对于包括规格、计划、能力、程序和指导方针的信息资产的计划和控制，来获取战略优势。它们通常由国家赞助商（或国家支持的实体）、犯罪团体或者竞争者控制实施。战术攻击通常是随机的或者投机取巧的；以恶意软件、漏洞、黑客、代理人、内部威胁和聊天室作为工具，对目标信息资产发动攻击以获得威望或者经济奖励，这些战术攻击多被专业黑客、脚本小子或者内部人士所操控。正如你看到的，战术攻击和战略攻击的核心不同点是动机：战术攻击是以网络资产为目标来获取威望和经济奖励，而战略攻击则是对于多个目标网络的战术攻击的协同合作（大规模地），以获得战略优势和抢占对手的先机。战术攻击的目标是随机和机会主义的，主要利用软件漏洞和用户的疏忽，而战略打击的目标则是被智能控制和精心安排的高等级过程。举例来说，战略攻击的目标包括对基础设施的渗透，对电信基础设施的攻击，搜集技术领域的特殊技术，比如隐形技术。要理解战略进攻就需要理解：（1）由个别网络支持的业务功能和流程；（2）网络之间的业务联系；（3）在承包商、供应商和目标实体之间共享战术攻击数据。对于这些业务关系的攻击所获取的信息用来指导和进行战略攻击。^[5]

针对用户的软件可以是战略性的、战术性的和投机取巧的。它们可能涉及攻击一个特定的用户，利用软件漏洞来获取通常不受用户限制的资源和信息，包括用户可以访问的资源或特定用户机器上的数据。战略攻击是杠杆战术和用户特定攻击的超级集合。

10.1.1 战略攻击

通常情况下，战略软件的目标是对于政府、经济和社会必不可少的基础设施功能的应用程序。关键基础设施的组件包括高速公路、机场与飞机、火车与铁路、公交系统、船舶、运输系统，基础物资发电厂、电力线的供应网络，以及各种油气管道和设施，这其中又包括供水和排污系统、土地和手机系统、计算机网络、电视和电台（不仅仅是公开的，还有在特殊网络中或特殊频率下被私人或政府实体控制的）、银行和其他金融机构、安全、消防、医院，以及应急服务机构。这些关键基础设施的每个组成元素都必不可少，即使是暂时地停止工作，也会对整个国家造成巨大的影响。即使某一个领域的基础设施受到威胁，其结果也是灾难性的。这其中包括电信、能源、交通、供水系统和应急服务系统。当然，战略攻击也包括对于政府部门必不可少的部分，包括国防、情报机构和其他被竞争对手认为具有极高价值的机构。

战略软件攻击具有高度的重复性和目标选择的一般性，如针对各行各业（军事、金融、能源等），也可以针对团体或者个人，同时战略攻击一定会产生持久的打击效果。战略攻击通

常没有战术攻击那般复杂，而且通常开发和维护成本更加低廉。这些类型的攻击可以归为三大领域：间谍、犯罪以及社会和政治。

10.1.2 战术攻击

网络战术威胁通常非常精确并且在技术上很复杂，同时有高度精确的目标。考虑到这种攻击战术的特殊本质，开发成本相当高。与战略攻击相比，战术攻击的重要性有所降低。在一些情况下，战术攻击可以用于增强攻击力或增强其他活动，如军事战役或者其他特殊利益行动小组的一个补充活动。考虑到战术攻击的精确性，它也可以应用到一系列的破坏活动中。鉴于战术攻击的成本，它们通常由资金充足的私人实体和政府提供资助，这些资助者往往是全球性受欢迎的国家、企业或特殊利益集团。

10.1.3 特定于用户的攻击

指定于用户的网络威胁本质上可以是战略性的，战术性的，或者针对个人，针对个人设备（要么是消费性的要么是企业所有的）。使用战略、战术或公开可用的方法来利用特定的个人或者金融、政治、私人利益的一般用户，将它们作为特定的目标，或作为到达另一个目标的一种手段，或用户随机破解的目标。

在许多方面，绝大多数的战略和战术攻击是一种用户攻击。这些攻击和特定于用户的攻击之间的差异在于其规模的不同。这种攻击的一个例子是，通过对其系统安装一个键盘记录器，以获得直接的经济效益（例如，得到其登录银行账户的密码），未经授权访问别人的电子邮件账户，或者目标是获取某测试的答案。所有这些攻击当中，只有少数人获利。这类攻击例子的主要目的是勒索，信用卡获利，为了获取（银行账户、社保号码等）针对特定的个人，未经授权访问社交媒体网站、电子信箱和其他在线信息，旨在勒索、利用个人，窃取个人信息，网络钓鱼和利用“智能家居”产品。读者将会熟悉其中大部分的攻击。勒索软件是一种恶意软件，它诱骗用户相信没有摆脱这些麻烦的办法，除了支付它。这种攻击的例子是，锁定用户的台式机，要求付费之后才能解锁。这种攻击最早在俄罗斯被发现，在近几年的时间里已传播到世界各地。

10.2 应用适当设计、管理和集中的 SDL 克服组织与业务挑战

我们概述了一个带有相关角色和责任的组织架构，这些角色和责任特定于在 SDL 模型里概述的任务，本书的作者已经实地检验和优化了这些任务。本书先前描述的结构能够有效地创建，实现和控制本书中的最佳范例，同时它也能够通过 SDL 模型中的 A1 ~ A5 来实现成功的买进和任务的管理。使用提到的组织架构有个附加的好处，就是你能够实现在第 8 章提到的发布后支持任务，这些任务通常不是你自己完成的，而是由别的组织完成。通过使用在 SDL 模型的每一部分提到过的指标，你不仅能够有效地管理和跟踪你的软件安全方案，而且能为公司管理、内部客户和方案的当前状态提供仪表盘。这个仪表盘也能够用来识别职员总数、资金和其他资源的缺口。最重要的是，通过内置安全性，你能够在发布后最大限度地避免发现软件中的安全漏洞。在某些情况下，当它们发生时，同时也能增加你成功管理这些安全漏洞的能力。

10.3 软件安全组织的现状和影响力

对于大多数组织来说，尽管采取基于工作量增加员工人数是常规计划，但是逐步增加员工总数对于本书提到的情况来说并不是合适的模型，更加不符合那些正在经历严峻考验的组织的具体情况。用更少的人做更多的事是我们一起面对的实际状况。为了解决这种两难的情况，我们为软件安全小组提出了一个并不依赖于连续增长或线性增长的模型。这个虚拟团队的人数并不是线性增长的，而是一个人员齐备、集中的软件安全小组，并且保持相对稳定。只要软件安全冠军项目基于本书，我们相信在你的软件工程开发组织中，一个由经验丰富的软件安全架构师组成的集中团队（每个安全架构师负责一个主要的软件产品）和负责一个软件产品的团队能够很好地按比例分配。另外，在一个开发组织中，通过共同承担一个常规产品安全事件响应小组（PSIRT）的责任，一个 PSIRT 经理应该能够满足整个组织中任务给定的共同承担责任。

如前所述，卓越并不意味着增加数量，而是你雇用的职工质量。每一个经验丰富的软件安全架构师能够协调和支持 SDL 在每个业务部和软件生产线的实施，也能够做到以下几方面。

- 为相关的软件安全冠军提供集中的软件安全小组处理和管理。
- 指导在安全性架构和安全性审查中的软件安全冠军。
- 在每一条软件产品生产线的指导下支持相关业务组的软件安全冠军。
- 为早期和及时的安全需求协调产品管理。
- 帮助计算项目安全风险。
- 帮助保证软件安全冠军创立合适和完整的安全性测试。
- 保证合适的安全性测试工具（静态、动态、模糊）可用于 SDL 中合适的地方。

当这些任务极大受益于老道的经验和判断力时，我们就有机会来节省这些高级技术领导者指导软件安全冠军和软件安全架构的成本。同时这这也是一个增加个体的参与度和节省公司与组织成本的好机会。某些有潜力通过经历和师徒制成为领导者的人是模型中软件安全冠军的一个极佳候选者。我们是我们所做一切的总和，而这个总和时常被修改和牢记。对于软件安全架构是一样的道理；这并不是一个时间点，而是一场旅行，并且需要不断学习，并且需要前辈的指导和帮助。

10.4 通过合理的政府管理克服 SDL 审计和法规挑战

第 2 章给出了 ISO/IEC 27034 的概述。除了书中提到的各种安全软件成熟度模型以外，这将是第一个软件安全标准。未来它将会有第三方认证并且认证行业将围绕它来组建。由于这个标准没有作为一个特定的国家安全标准定义应用程序，而是作为一个过程定义它，组织可以进行应用控制和测量，以管理使用的风险，我们相信我们的模型是符合本标准的。这个标准为把安全整合到管理应用程序的过程中提供了准则在详细说明、设计、开发、测试、实现和维护应用系统中的安全功能和控制过程中，需要一个明确的实现过程。ISO/IEC 27034 里面特定的要求和过程并没有要求隔离实现，但是需要集成到一个组织现有的过程中。将 ISO/IEC 27034 准则与 SDL 实践或者书中先前提到的成熟模型整合，将会让你自如地应对审计、监管或者管理的挑战，因为坚持规范将可能由后者驱动的指导来驱动。

10.5 软件安全的未来预测

我们在这里把这一节成两部分。首先，坏消息是我们看到的事情很可能会在工业界普及，但有些内容应该改变；其次，软件安全是好东西，如果你愿意，未来非常光明。

10.5.1 坏消息

先说坏消息。在大多数情况下，除了威胁建模和架构安全审查之外，这是一门艺术，而不是一门科学，软件安全并不难，它是一个很多年前就已知领域，但没有被深入研究、挖掘。一个典型的表现是在公共漏洞与披露（CVE）和 OWASP 以及 SAN 的前 10 项漏洞列表，10 年来都没有变化过。虽然在过去的几年中业界在这个领域保持领先，而且 ISO ISO / IEC 27034、29147、30111 已经声称，他们在可预见的未来把软件安全视为一个持续的问题。虽然未来在这方面是光明的，但是最终引导业界朝着正确的方向前进需要时间。如本书所述，把安全内置到软件开发过程中更多的是一种态度的转变，管理验收和业务 / 运营过程变化，而不是新的科技或技术学科中的灵光一现。

漏洞的后修复期代价是非常高的。调整 and 保持现有产品安全性的工具比购买新产品更贵。虽然对产品进行更改的责任在于供应商，我们还是有一些可能有助于改变这种现状的办法。我们提出了一个从软件安全性中移除漏洞的范例。然而，并不是每个漏洞都可以被利用或者被攻击。通常情况下，减少了那些不明显的一般漏洞扫描器，甚至使得攻击者对普通漏洞丧失了兴趣。我们不是说我们停止修正代码中的 bug。恰恰相反，如本书所述，尽管报告了数千个漏洞并没有使软件安全。作为一个整体，安全行业将继续专注于漏洞：每一个新的攻击类型，每一个新的变化，以及每一个可以想象的攻击方式。相反，专注于正确的程序行为很符合开发商的做法设计和代码创建。根据我们的经验，开发人员会因于正确性得到奖励。显而易见，漏洞是错误，这很简单明了。但是，关注正确性在软件安全中是突发性转变。今天的工具往往不报告单一的 bug，这个 bug 能对一个特定类型的攻击的多种变化作出反应。相反，很多时候，工具报告的每个漏洞变体需要由开发商来解决。这才是安全人士需要考虑的问题。这是一个攻击者的看法：哪些攻击方法将针对这个特定的系统实施攻击？时至今日（写作本书时），大多数漏洞扫描仪都能解决这个问题。然而，写代码的人只是想知道编码错误是什么，它在代码中的什么地方，什么是必须编程的正确行为。通常情况下，如果该工具包含任何编程提示，这些往往被埋在该工具的用户界面之下。相反，工具应该不会比任何一个编译器更难使用。其结果可能是一个编码错误列表其中包含错误的行号。它所用的代码片段指出其中的错误所在。当然，这是一种过于简单化的方法。某些类型的安全漏洞位于整个代码段，或甚至可以跨整个系统。不过，焦点是什么才是编码错误，什么是它的解决方案。逻辑错误可能会从设计解决方案方面描述：妥善随机会话 ID，或包括不可预知的会话标识符与每个 Web 输入（例如，防止跨站点请求伪造）等。当今世界，数以千万计的人正在编写 Web 代码，而大量的代码含有可利用的漏洞，我们需要简化的方法来发现这些错误代码。数以百万计的庞大数量漏洞没有减少攻击面。我们想建议要求使用这种新方法——“以开发人员为中心的软件安全性设计”。那意味着，安全人员应该了解开发者的重点与开发者的问题。安全行业必须着手解决这些以在应有的位置考虑安全性，熟练的程序员也必须考虑可维护性、算法的正确性、计算的正确性，以及所有其他问题。

10.5.2 好消息

正如本书通篇和上一节所提到的那样，安全行业知道该怎么做，也知道他们应该这样做，以及如何做到这一点，但他们不做。知道做什么是需要取得胜利的斗争的关键，我们认为来自新的 ISO 生产标准（27034、29147、30111 和）的压力和最近企业和政府团体安全意识的增加（内置在软件开发过程中的有效动力），最终会使软件安全性拥有高优先级并推动业务发展。另一个好消息是工具和软件安全培训不断提高。我们也看到了下一代软件架构师越来越多的指导，随着时间的推移，这将成为我们这个行业的强劲动力。最重要的是，就像这本书所描述的，新的组织和管理的 SDL 模型（基于现实经验和成功）正在开发中。

10.6 总结

软件安全的重要性是不容小觑的，因为我们很快就走向这个新的时代，以前的任务转交给给人脑，现在正被由软件驱动的机器所取代。正是出于这个原因，我们写了这本书。与此相反，在可预见的将来，人们将会继续编写软件程序。这也意味着，新的软件将继续在软件安全得到重视或复杂的攻击变得普遍之前所编写的遗留代码或软件上进行构建。只要我们编写程序，成功达成软件安全性的关键是使软件开发项目过程更加高效和有效。虽然本书的方法包括使软件安全的人员、流程和技术方法，但是作者始终认为对于软件安全性，人仍然是管理中最重要的重要组成部分。只要软件的开发、管理和利用都是通过人实现的，这个观点就一直是正确的。本书对软件安全性提出了一个有关当下技术、运营、业务和开发环境的具体流程。我们专注于人们可以通过最佳做法和度量标准的形式来做什么去控制和管理一个安全的软件开发过程。虽然安全性不是最近几年业界开发软件的自然组成部分，但作者始终认为，提高软件安全性对开发流程是可行的、实用的和必要的。我们相信，本书介绍的软件安全最佳实践和模式将会使所有阅读这本书的人明白这一点，包括高管、项目经理和从业人员。

当谈到网络安全时，我们相信这是关于软件本身以及它是否是安全的，因此这本书的英文书名是 Core Software Security: Security at the Source。你可以拥有世界上最好的客户端、主机和网络安全性，包括加密传输和数据存储，但如果应用软件存在漏洞，并且可以被利用，你的纵深防御安全方法就必然会遭到削弱。正如老话所说，你最薄弱的环节决定你综合能力的高低，而在今天的世界上，这一环仍然是软件；并且软件渗透到我们全部的生活中，国防、医学、工业、金融、农业、交通运输以及我们如何管理自己和生活。这是一件非常严肃的事，并且它还有着令人生畏的漏洞。你只需要去看看多少年来相同的软件漏洞仍然存在于 CVE 前 25 或 OWAS 和 PSAN 前 10 的排名中，就能认识到，企业仍然没有重视软件安全。更糟糕的是，经验丰富和专业的对手将针对存在漏洞的软件，并且不一定要要求它处于互联网链接的状态下才会有威胁，尽管这能让攻击变得更容易，但软件仍是首要目标，因为如果你可以拥有软件，就表示你可以拥有由它控制的数据和流程。在今天的世界上，这可能会导致生命威胁和严重的区域性或全球性的后果。本书已经介绍了 SDL 的最佳实践和度量标准来优化安全软件开发生命周期的搭建、管理和发展，以及可最大限度地缓解此类风险的编程。管理软件的安全性是作者每天所面对的工作，而本书正是基于我们的实际经验。我们一直与“财富 500 强”企业合作，经常看到不遵循安全开发生命周期（SDL）实践的例子。在本书中，我们采取了经验为基础的方法来运用现有最佳 SDL 模型的组件，去处理上述以 SDL 软件安全最佳实

践模型和框架的形式描述的问题。最重要的是,SDL 最佳实践模型已被映射到软件开发生命周期的标准模型之中,它详细解释了如何使用它来建立和管理一个成熟的 SDL 程序。尽管安全问题会一直存在,但是本书旨在教你如何在软件面市之前最大限度地发挥一个团队的能力,通过将安全性内置在开发过程之中,来最大限度地减少软件产品的漏洞。我们希望你喜欢这本书,享受到我们在编写过程中所有的乐趣,因为我们致力于通过我们的努力,帮助减轻全世界范围内,尤其是读者所面对的软件漏洞风险。

参考文献

1. Snyder, J. (2012). "Growing Cyber Threats Demand Advanced Mitigation Methodologies." Retrieved from http://www.raytheon.com/capabilities/rtnwcm/groups/iis/documents/content/rtn_iis_cyber_whitepaper_wcs.pdf.
2. Ellison, R. (2006). "Security and Project Management." Retrieved from <https://buildsecurityin.us-cert.gov/articles/best-practices/project-management/security-and-project-management>.
3. Acohido, B. (2013, February 27). "Former Cybersecurity Czar Pursues Safer Software." Retrieved from <http://www.usatoday.com/story/tech/2013/02/27/howard-schmidt-executive-director-safecode/1952359>.
4. Ponemon Institute and Security Innovation (2013, August 27). *The State of Application Security—A Research Study by Ponemon Institute LLC and Security Innovation*, p. 21). Retrieved from <https://www.securityinnovation.com/uploads/ponemon-state-of-application-security-maturity.pdf>.
5. Gilbert, L., Morgan, R., and Keen, A. (2009, May 5). "Tactical and Strategic Attack Detection and Prediction," U.S. Patent 7530105. Retrieved from <http://www.freepatentsonline.com/7530105.html>.
6. Encyclopedia of Espionage, Intelligence, and Security (2013). *Espionage Encyclopedia: Critical Infrastructure*. Retrieved from <http://www.faqs.org/espionage/Cou-De/Critical-Infrastructure.html>.
7. Linktv.org (2013). *Cyber Espionage*. Retrieved from <http://news.linktv.org/topics/cyber-espionage>.
8. U.S.—China Economic and Security Review (2012). *2012 Report to Congress of the U.S.—China Economic and Security Review Commission—One Hundred Twelfth Congress—Second Session*. Retrieved from http://origin.www.uscc.gov/sites/default/files/annual_reports/2012-Report-to-Congress.pdf.
9. Information Warfare Monitor (2009). *Tracking GhostNet: Investigating a Cyber Espionage Network*. Retrieved from <http://www.scribd.com/doc/13731776/Tracking-GhostNetInvestigating-a-Cyber-Espionage-Network>.
10. McAfee Labs and McAfee Foundstone Professional Services (2010). *Protecting Your Critical Assets—Lessons Learned from "Operation Aurora"*. Retrieved from <http://www.mcafee.com/us/resources/white-papers/wp-protecting-critical-assets.pdf>.
11. Nakashima, E. (2011, August 02). "Report on 'Operation Shady RAT' Identifies Widespread Cyber-Spying." *The Washington Post*. Retrieved from http://articles.washingtonpost.com/2011-08-02/national/35269748_1_intrusions-mcafee-china-issues.
12. Symantec (2011, August 4). "The Truth Behind the Shady Rat." *Symantec Official Blog*. Retrieved from <http://www.symantec.com/connect/blogs/truth-behind-shady-rat>.
13. Hsu, T. (2011, February 10). "China-Based Hackers Targeted Oil, Energy

- Companies in 'Night Dragon' Cyber Attacks, McAfee Says." *Los Angeles Times*. Retrieved from <http://latimesblogs.latimes.com/technology/2011/02/chinese-hackers-targeted-oil-companies-in-cyberattack-mcafee-says.html#sthash.d7PrG6Iy.dpuf>.
14. McAfee Foundstone Professional Services and McAfee Labs (2011, February 10). *Global Energy Cyberattacks: "Night Dragon."* Retrieved from <http://www.mcafee.com/us/resources/white-papers/wp-global-energy-cyberattacks-night-dragon.pdf>.
 15. Graham, B. (2005, August 25). "Hackers Attack via Chinese Web Sites." *The Washington Post*. Retrieved from <http://www.washingtonpost.com/wp-dyn/content/article/2005/08/24/AR2005082402318.html>.
 16. Thornburgh, N. (2005, August 25). "Inside the Chinese Hack Attack." *Time Magazine*. Retrieved from <http://content.time.com/time/nation/article/0,8599,1098371,00.html>.
 17. Onley, D. and Wait, P. (2006, August 17). "Red Storm Rising." *GCN*. Retrieved from <http://gcn.com/Articles/2006/08/17/Red-storm-rising.aspx?Page=2&p=1>.
 18. Sandlin, S. (2007, October 14). "Analyst, Sandia Settle Suit." *Albuquerque Journal*. Retrieved from <http://www.abqjournal.com/news/metro/602547metro10-14-07.htm>.
 19. Interpol (2013). *Cybercrime*. Retrieved from <http://www.interpol.int/Crime-areas/Cybercrime/Cybercrime>.
 20. Williams, P. (2013). *Organized Crime and Cyber-Crime: Implications for Business*. CERT Coordination Center (CERT/CC). Retrieved from www.cert.org/archive/pdf/cybercrime-business.pdf.
 21. Williams, P. (2013). *Organized Crime and Cybercrime: Synergies, Trends, and Responses*. Retrieved from <http://www.crime-research.org/library/Cybercrime.htm>.
 22. Samani, R., and Paget, F. (2011). *Cybercrime Exposed—Cybercrime-as-a-Service, McAfee—An Intel Company White Paper*. Retrieved from <http://www.mcafee.com/us/resources/white-papers/wp-cybercrime-exposed.pdf>.
 23. Gandhi, R., Sharma, A., Mahoney, W., Sousan, W., Zhu, Q., and Laplante, P. (2011, February). *Dimensions of Cyber-Attacks: Cultural, Social, Economic, and Political*. ResearchGate.net, Source: IEEE Xplore. Retrieved from http://www.researchgate.net/publication/224223630_Dimensions_of_Cyber-Attacks_Cultural_Social_Economic_and_Political.
 24. Vaughan-Nichols, S. (2012, January 20). "How Anonymous Took Down the DoJ, RIAA, MPAA and Universal Music Websites." *ZDNet*. Retrieved from <http://www.zdnet.com/blog/networking/how-anonymous-took-down-the-doj-riaa-mpaa-and-universal-music-websites/1932>.
 25. Tucker, N. (2008, January 18). "Tom Cruise's Scary Movie; In Church Promo, the Scientologist Is Hard to Suppress." *The Washington Post*. Retrieved from <http://www.highbeam.com/doc/1P2-15129123.html>.
 26. *The Economist* (2008, February 2). "Fair Game; Scientology. (Cyberwarfare Against a Cult) (Anonymous)." Retrieved from <http://www.highbeam.com/doc/1G1-174076065.html>.
 27. BBC (2010, December 9). "Anonymous Hacktivists Say Wikileaks War to Continue." Retrieved from <http://www.bbc.co.uk/news/technology-11935539>.
 28. Swallow, E. (2011, August 14). "Anonymous Hackers Attack BART Website." *Mashable*. Retrieved from <http://mashable.com/2011/08/15/bart-anonymous-attack>.
 29. Kingsbury, A. (2010, April 14). "Documents Reveal Al Qaeda Cyberattacks—The Attacks Were Relatively Minor but Show the Group's Interest in Cyberwar." *U.S. News & World Report*. Retrieved from <http://www.usnews.com/news/>

- articles/2010/04/14/documents-reveal-al-qaeda-cyberattacks.
30. Tiller, J. (2010, June 10). "Cyberwarfare: It's a New Theater of War, Not Just a New Form of War." *Real Security*. Retrieved from <http://www.realsecurity.us/weblog/?e=104>.
 31. Clarke, R. A. (2010). *Cyber War*. HarperCollins, New York.
 32. The Economist, (2010, July 1). "Cyberwar: War in the Fifth Domain." *The Economist*.
 33. Lynn, W. J., III. (2010, Sept./Oct.) "Defending a New Domain: The Pentagon's Cyberstrategy." *Foreign Affairs*, pp. 97–108.
 34. U.S. Department of Defense Strategy for Operating in Cyberspace, July 2011, p. 3.
 35. Herzog, S. (2011, Summer). "Revisiting the Estonian Cyber Attacks: Digital Threats and Multinational Responses." *Journal of Strategic Security*, Vol. 4, No. 2, Strategic Security in the Cyber Age, Article 4. Retrieved from <http://scholarcommons.usf.edu/cgi/viewcontent.cgi?article=1105&context=jss>.
 36. RIANOVOSTI (2007, September 6). "Estonia Has No Evidence of Kremlin Involvement in Cyber Attacks." Retrieved from <http://en.rian.ru/world/20070906/76959190.html>.
 37. Rehman, S. (2013, January 14). "Estonia's Lessons in Cyberwarfare." *U.S. News Weekly*. Retrieved from <http://www.usnews.com/opinion/blogs/world-report/2013/01/14/estonia-shows-how-to-build-a-defense-against-cyberwarfare>.
 38. Swaine, J. (2008, August 11). "Georgia: Russia 'Conducting Cyber War.'" *The Telegraph*. Retrieved from <http://www.telegraph.co.uk/news/worldnews/europe/georgia/2539157/Georgia-Russia-conducting-cyber-war.html>.
 39. Falliere, N., Murchu, L., and Chien, E. (2011, February). *W32.Stuxnet Dossier, Version 1.4—Symantec Security Response*. Retrieved from http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/w32_stuxnet_dossier.pdf.
 40. Schneier, B. (2010, October 7). "Stuxnet." *Schneier on Security—A Blog Covering Security and Security Technology*. Retrieved from <https://www.schneier.com/blog/archives/2010/10/stuxnet.html>.
 41. Dunn, J. (2012, March 9). "Ransom Trojans Spreading Beyond Russian Heartland: Security Companies Starting to See More Infections." *Techworld*. Retrieved from <http://news.techworld.com/security/3343528/ransom-trojans-spreading-beyond-russian-heartland>.

关键的成功因素、可交付成果、SDL 模型每个阶段的指标

第 3 章到第 7 章已经概述了关键的成功因素、可交付成果和应该作为安全开发生命周期 (SDL) 模型一部分的指标。第 8 章概括了关键的可交付成果和指标。关键的成功因素、可交付成果和指标并不是一成不变的，它们需要进行调整，以便把 SDL 映射到软件开发生命周期 (SDLC) 中。本附录总结从第 3 章到第 8 章概括的关键成功因素、可交付成果和指标。

表 A-1 SDL 每个阶段的关键成功因素

阶段	关键成功因素	描述
安全评估 (A1): SDL 活动与最佳实践	1. 计划 SDL 活动的准确度	所有 SDL 活动被准确地识别
	2. 产品风险轮廓	管理部门了解开发产品的真实成本
	3. 威胁轮廓的准确度	环境中产品成功的缓解措施和对策很到位
	4. 有关规定、认证和法规遵从框架的覆盖	所有适用的法律和法规遵从方面都被覆盖
	5. 软件所需安全目标的覆盖	“必须有”的安全目标需要满足
架构 (A2): SDL 活动与最佳实践	1. 确定业务需求和风险	按 CIA 定义的业务需求和风险的映射
	2. 有效的威胁建模	识别软件威胁
	3. 有效的架构威胁分析	分析软件威胁和威胁出现的概率
	4. 有效的风险缓解策略	每一个业务需求的风险接受、容忍和缓解计划
	5. DFD 的准确度	威胁建模中使用的数据流图
设计和开发 (A3): SDL 活动与最佳实践	1. 全面的安全测试计划	映射 SDLC 不同阶段需要的安全测试类型
	2. 有效的威胁建模	识别软件威胁
	3. 设计安全分析	不同软件组件的威胁分析
	4. 隐私实施评估	根据评估实施隐私相关控制所需要的工作量
	5. 策略一致性审查 (更新)	涉及阶段 3 的策略遵从更新
设计和开发 (A4): SDL 活动与最佳实践	1. 安全测试用例执行	覆盖所有相关的测试用例
	2. 安全测试	完成所有类型的安全测试，修复找到的问题
	3. 隐私验证和修复	隐私相关控制的有效性，修复找到的问题
	4. 策略合规性检查	按照阶段 4 更新策略
发布 (A5): SDL 活动与最佳实践	1. 策略一致性分析	开发阶段中的最终安全审查和一致性需求
	2. 安全漏洞扫描	扫描软件栈确认安全问题
	3. 渗透测试	利用所有软件栈上的安全问题
	4. 开源许可审查	最终审查在栈中使用的开源软件
	5. 最终安全审查	最终审查 SDL 周期中确认的安全需求的一致性
	6. 最终隐私审查	最终审查 SDL 周期中所有隐私需求的一致性
	7. 客户协议框架	确认与客户共享的安全信息的框架

表 A-2 SDL 每个阶段的可交付成果

阶段	可交付成果	目标
安全评估（A1）：SDL 活动与最佳实践	产品风险轮廓	预估产品的真实成本
	SDL 项目概述	将 SDL 映射到开发进度
	适用的法律和法规	在适用法律下获得利益相关者的正式批准
	威胁轮廓	引导 SDL 活动以减轻威胁
	认证要求	列出产品和业务认证的要求
	第三方软件列表	识别第三方软件的依赖
架构（A2）：SDL 活动与最佳实践	度量标准模板	建立定期向主管汇报的节奏
	业务需求	软件需求，包括 CIA
	威胁建模工件	数据流图、元素、威胁列表
	架构威胁分析	基于威胁分析的威胁和风险的优先级
	风险缓解计划	计划去缓解、接受或者容忍风险
设计和开发（A3）：SDL 活动与最佳实践	政策符合性分析	遵守公司政策的分析
	更新的威胁建模标准	数据流图、要素、威胁列表
	设计安全审查	基于安全评估的软件组件设计修改
	安全测试计划	计划减轻、接受或容忍风险
	更新的策略遵从分析	坚持遵守公司政策
设计和开发（A4）：SDL 活动与最佳实践	隐私实施评估结果	隐私评估的建议
	安全测试执行报告	检查安全测试用例执行的进程
	更新策略的合规性分析	分析公司策略的遵守程度
	隐私遵守报告	验证执行的隐私评估是否符合建议
	安全测试报告	不同类型的安全测试发现的结果
发布（A5）：SDL 活动与最佳实践	修复报告	产品的安全状态
	更新策略一致性分析	分析公司策略
	安全测试报告	SDL 的这个阶段找到的不同类型的安全检查结果
	修复报告	提供安全修复报告
	开源许可审查报告	对开源软件许可需求一致性的检查
	最终安全和隐私审查报告	检查安全和隐私需求的一致性
发布后的支持（PRSA1-5）	客户协议框架	在产品生命周期的不同阶段对应参与用户详细的框架
	外部漏洞信息披露响应过程	定义安全漏洞评估和沟通的过程
	发布后认证	来自外部各方的认证，展示产品 / 服务的安全状态
	第三方安全审查	除内部测试团队以外的团队进行的安全评估
	遗留代码、M&A 和 EOL 计划的安全策略与过程	减轻遗留代码和 M&As 带来的安全风险的策略

表 A-3 SDL 每个过程的指标

阶 段	指 标
安全评估 (A1): SDL 活动与最佳实践	软件安全团队循环的时间 (单位: 周)
	参加 SDL 的利益相关者的百分比
	SDL 活动映射到开发活动的百分比
	安全目标满足的百分比
架构 (A2): SDL 活动与最佳实践	业务威胁、技术威胁 (映射到业务威胁) 和威胁参与者的列表
	这个阶段之后不满足的安全目标个数
	遵守公司 (现有) 政策的百分比
	软件入口点的数量 (使用的 DFD)
	风险 (和威胁) 接受、减轻和容忍的百分比
	重新定义的初始软件需求百分比
	产品中计划的软件体系结构 (主要和次要) 的变化数量
	根据安全要求所需的软件体系结构更改数量
设计和开发 (A3): SDL 活动与最佳实践	威胁、可能性和严重性
	符合公司政策的百分比 (更新)
	符合第 2 阶段和第 3 阶段的百分比
	软件的切入点 (使用 DFD)
	接受的风险相对于缓解的风险的百分比
	最初的软件重新定义要求的百分比
	软件架构变化的百分比
	没有响应的软件安全测试的 SDLC 阶段的百分比
	与隐私控制有关实现的软件组件的百分比
	代码行数
	使用静态分析工具发现的安全缺陷数量
	使用静态分析工具发现的高风险缺陷数
设计和开发 (A4): SDL 活动与最佳实践	缺陷密度 (每 1000 行代码中的安全问题)
	遵守公司策略的比例 (更新)
	• 遵守阶段 3 和阶段 4 的比例对比
	通过静态测试工具实际测试的代码行数
	通过静态分析工具找到的安全缺陷数量
	通过静态分析工具找到的高风险安全缺陷的数量
	缺陷密度 (每一千行代码的安全问题数量)
	通过静态分析、动态分析、人工代码审查、渗透测试、模糊测试找到的安全问题的类型和数量
	• 使用不同类型的交叉测试找出的安全问题
	• 对比不同测试类型找到的安全问题的严重程度
	• 把找到的安全问题与之前确定的威胁 / 风险进行映射
	对发现的安全问题进行修复的数量
	• 发现问题的严重程度
	• 花费在修复发现问题上的小时数 (大约)
	发现问题中未解决问题的数量、类型和严重程度
	遵守测试安全计划的比例

(续)

阶 段	指 标
设计和开发 (A4): SDL 活动与最佳实践	安全测试用例执行的数量
	<ul style="list-style-type: none">• 执行安全测试用例发现的问题数量• 执行回归测试的数量
发布 (A5): SDL 活动与最佳实践	遵守公司策略的百分比 (更新的)
	<ul style="list-style-type: none">• 阶段 4 与 5 中遵守的百分比
	安全漏洞扫描和渗透测试发现的安全问题的数量、类型和严重性
	<ul style="list-style-type: none">• 不同类型测试中找到的交叠的安全问题• 不同类型测试发现的安全问题严重性的对比• 发现的安全问题与之前确认的威胁 / 风险的映射
	发现的安全问题的修复数量 (更新的)
	<ul style="list-style-type: none">• 发现的问题的严重性• 修复发现的问题花费的小时数
	发现的严重问题的数量、类型、严重性 (更新的)
发布后支持 (PRSA1-5)	遵守安全和隐私需求的百分比
	应对外部披露的安全漏洞的时间 (以小时为单位)
	每月全职员工 (Full-Time Employee, FTE) 处理对外披露过程需要的小时数
	产品发布后发现的安全问题的数目 (按严重程度排列)
	每月客户报告的安全问题数目
	任意 SDL 活动中, 客户报告的安全问题没有确定的数目

软件安全从源头开始

Core Software Security Security at the Source

这本引人入胜的书使得读者可以在各种规模的软件开发和工程组织中构建安全的产品。本书向高管人员阐述了关于软件安全性应该做出的决策，并向经理和开发人员提供过程方面的指导，使读者可以应用牢固的解决方案来抵制网络威胁。

—— Dena Haritos Tsamitis博士，卡内基梅隆大学信息网络学院主任和CyLab教育主任

本书是软件安全专家的权威指南。两位作者巧妙地勾勒出把安全性整合到安全软件开发过程中的程序和策略，以及为什么安全需要以软件和相关开发人员为中心……在网络战争最前线的人（特别是软件开发人员和相关同事）必须具备安全意识。

—— Colonel Cedric Leighton，美国退役空军，Cedric Leighton Associates创始人&总裁

在云计算和移动应用程序兴起之后，软件安全问题从来没有比现在更重要。本书是安全专家、软件开发人员和软件工程师的必读书目。作者出色地提供了通用的方法，以实现强大的软件安全态势。

—— Larry Ponemon博士，Ponemon研究所主席&创始人

软件安全的根源在于软件开发人员开发的源代码。因此，安全应该以开发人员为中心，注重源代码的安全开发。本书揭示了将安全构建到整个软件开发生命周期中的方法和过程，使得软件在源头就是安全的！

—— Eric S. Yuan，Zoom视频通信有限公司创始人&CEO

作者简介

詹姆斯·兰萨姆 (James Ransome) 产品安全方面的高级总监，全面负责McAfee产品安全项目。他职业生涯的起点是在私人 and 公共行业担任领导职务，其中包括三个首席信息安全官 (CISO) 和四个首席安全官 (CSO) 的职务。在进入企业领域前，他有23年服务于政府，包括支持美国情报界、联邦执法和国防部的各种角色。James拥有信息系统的博士学位，是多本信息安全书籍的作者。James是计算机与信息学科国际荣誉协会Upsilon Pi Epsilon的成员，同时还是注册信息安全经理 (CISM)、信息系统安全认证专家 (CISSP) 和波耐蒙研究所 (Ponemon Institute) 的杰出研究员。

安莫尔·米斯拉 (Anmol Misra) 信息安全专家和资深安全专业人士。他的专长包括移动和应用安全、漏洞管理、应用和基础设施的安全评估和安全代码审查。他在思科公司信息安全组担任项目经理，主要负责制定与实施安全策略和方案，以推动安全最佳实践纳入到思科主导的产品的各个方面。加入思科之前，Anmol是安永会计事务所的高级顾问，给财富500强客户提供定义和改进信息安全计划和实践的咨询服务。他帮助企业降低IT安全风险，并通过改善其安全状况使其遵从法规。Anmol拥有卡内基梅隆大学信息网络硕士学位和计算机工程工学学士学位。



投稿热线: (010) 88379604
客服热线: (010) 88378991 88361066
购书热线: (010) 68326294 88379649 68995259

华章网站: www.hzbook.com
网上购书: www.china-pub.com
数字阅读: www.hzmedia.com.cn



上架指导: 计算机/安全

ISBN 978-7-111-54023-6



9 787111 540236 >

定价: 69.00元